Chapter 5 B
Large and Fast: Exploiting Memory Hierarchy
Dependability
Dependability

- Fault: failure of a component
  - May or may not lead to system failure

Service accomplishment
Service delivered as specified

Restoration
Failure

Service interruption
Deviation from specified service
Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = $\frac{MTTF}{MTTF + MTTR}$
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair
The Hamming SEC Code

- Hamming distance
  - Number of bits that are different between two bit patterns
- Minimum distance = 2 provides single bit error detection
  - E.g. parity code
  - Use white board for illustration (3+1 bits).
- Minimum distance = 3 provides single error correction, 2 bit error detection
Encoding SEC

- To calculate Hamming code:
  - Number bits from 1 on the left
  - All bit positions that are a power 2 are parity bits
  - Each parity bit checks certain data bits (to add even parity)

<table>
<thead>
<tr>
<th>Bit position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoded date bits</td>
<td>p1</td>
<td>p2</td>
<td>d1</td>
<td>p4</td>
<td>d2</td>
<td>d3</td>
<td>d4</td>
<td>p8</td>
<td>d5</td>
<td>d6</td>
<td>d7</td>
<td>d8</td>
</tr>
<tr>
<td>Parity bit coverate</td>
<td>p1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 6
Decoding SEC

- Value of parity bits indicates which bits are in error
  - Use numbering from encoding procedure
  - E.g. Syndrome in (P8,P4,P2,P1)
  - In Syndrome 0= OK, 1= error
    - Parity bits Syndrome = 0000 indicates no error
    - Parity bits Syndrome = 1010 indicates bit 10 was flipped
Hamming Code: Encoding

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoded Data Bits</td>
<td>P1</td>
<td>P2</td>
<td>D1</td>
<td>P4</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>P8</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
</tr>
<tr>
<td>P1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>P2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>P4</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>P8</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Example Data</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Example parity</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Encoded word</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

P1 provides even parity (0) for \((D1, D2, D4, D5, D7) = (1, 0, 1, 1, 1)\)
## Hamming Code: Checking

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Encoded Data</strong></td>
<td><strong>P1</strong></td>
<td><strong>P2</strong></td>
<td><strong>D1</strong></td>
<td><strong>P4</strong></td>
<td><strong>D2</strong></td>
<td><strong>D3</strong></td>
<td><strong>D4</strong></td>
<td><strong>P8</strong></td>
<td><strong>D5</strong></td>
<td><strong>D6</strong></td>
<td><strong>D7</strong></td>
<td><strong>D8</strong></td>
</tr>
<tr>
<td><strong>P1 even, OK</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>P2 Odd, error</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>P4 even, ok</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>P8 Odd, error</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>Encoded word</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

P1 (0) for (D1,D2,D4,D5, D7) = (1,0,1,1,1)  Parity even, Syndrome bit 0
P2 (1) for (D1,D3,D4,D6, D7) = (1,0,1,1,1)  Parity not even, Syndrome bit 1
Complete syndrome (P8,P4,P2,P1) = (1010). Error in bit 1010<sub>binary</sub>=10
Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
- Managed jointly by CPU hardware and the operating system (OS)
- Address mapping:
  - Virtual address space used by the processor
  - Real address space used by memory
  - How to implement mapping efficiently?
Virtual Memory

- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault
Address Translation

- Fixed-size pages (e.g., 4K)
Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code

- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms
Page Tables

- Stores placement information
  - Array of page table entries (PTEs), indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk
Translation Using a Page Table

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 15
Mapping Pages to Storage

Virtual page number

Page table
Valid
Physical page or disk address

Physical memory

Disk storage

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 16
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently

- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written
Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software
Fast Translation Using a TLB

Virtual page number | Valid Dirty Ref | Physical page number
--- | --- | ---
1 0 1 | | 
1 1 1 | | 
1 1 1 | | 
1 0 1 | | 
0 0 0 | | 
1 0 1 | |

Page table

Valid Dirty Ref | Physical page or disk address
--- | ---
1 0 1 | 
1 0 0 | 
1 0 0 | 
1 0 1 | 
1 0 1 | 
0 0 0 | 
1 0 1 | 
1 0 1 | 
1 0 1 | 
0 0 0 | 
1 1 1 | 
1 1 1 | 
1 1 1 | 
0 0 0 | 
1 1 1 | 

TLB

Physical memory

Disk storage

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 19
TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler

- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction
TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not present

- Must recognize TLB miss before destination register overwritten
  - Raise exception

- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur
Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction
TLB and Cache Interaction

- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address
Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance

- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)
The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching

- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy
# Multilevel On-Chip Caches

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>ARM Cortex-A8</th>
<th>Intel Nehalem</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache organization</td>
<td>Split instruction and data</td>
<td>Split instruction and data</td>
</tr>
<tr>
<td></td>
<td>caches</td>
<td>caches</td>
</tr>
<tr>
<td>L1 cache size</td>
<td>32 KiB each for instructions/data</td>
<td>32 KiB each for instructions/data per core</td>
</tr>
<tr>
<td>L1 cache associativity</td>
<td>4-way (I), 4-way (D) set associative</td>
<td>4-way (I), 8-way (D) set associative</td>
</tr>
<tr>
<td>L1 replacement</td>
<td>Random</td>
<td>Approximated LRU</td>
</tr>
<tr>
<td>L1 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L1 write policy</td>
<td>Write-back, Write-allocate(?)</td>
<td>Write-back, No-write-allocate</td>
</tr>
<tr>
<td>L1 hit time (load-use)</td>
<td>1 clock cycle</td>
<td>4 clock cycles, pipelined</td>
</tr>
<tr>
<td>L2 cache organization</td>
<td>Unified (instruction and data)</td>
<td>Unified (instruction and data) per core</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>128 KiB to 1 MiB</td>
<td>256 KiB (0.25 MiB)</td>
</tr>
<tr>
<td>L2 cache associativity</td>
<td>8-way set associative</td>
<td>8-way set associative</td>
</tr>
<tr>
<td>L2 replacement</td>
<td>Random(?)</td>
<td>Approximated LRU</td>
</tr>
<tr>
<td>L2 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L2 write policy</td>
<td>Write-back, Write-allocate (?)</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L2 hit time</td>
<td>11 clock cycles</td>
<td>10 clock cycles</td>
</tr>
<tr>
<td>L3 cache organization</td>
<td>–</td>
<td>Unified (instruction and data)</td>
</tr>
<tr>
<td>L3 cache size</td>
<td>–</td>
<td>8 MiB, shared</td>
</tr>
<tr>
<td>L3 cache associativity</td>
<td>–</td>
<td>16-way set associative</td>
</tr>
<tr>
<td>L3 replacement</td>
<td>–</td>
<td>Approximated LRU</td>
</tr>
<tr>
<td>L3 block size</td>
<td>–</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L3 write policy</td>
<td>–</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L3 hit time</td>
<td>–</td>
<td>35 clock cycles</td>
</tr>
</tbody>
</table>
# 2-Level TLB Organization

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>ARM Cortex-A8</th>
<th>Intel Core i7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual address</td>
<td>32 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td>Physical address</td>
<td>32 bits</td>
<td>44 bits</td>
</tr>
<tr>
<td>Page size</td>
<td>Variable: 4, 16, 64 KiB, 1, 16 MiB</td>
<td>Variable: 4 KiB, 2/4 MiB</td>
</tr>
<tr>
<td>TLB organization</td>
<td>1 TLB for instructions and 1 TLB for data</td>
<td>1 TLB for instructions and 1 TLB for data per core</td>
</tr>
<tr>
<td></td>
<td>Both TLBs are fully associative, with 32 entries, round robin replacement</td>
<td>Both L1 TLBs are four-way set associative, LRU replacement</td>
</tr>
<tr>
<td></td>
<td>TLB misses handled in hardware</td>
<td>L1 I-TLB has 128 entries for small pages, 7 per thread for large pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L1 D-TLB has 64 entries for small pages, 32 for large pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L2 TLB is four-way set associative, LRU replacement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L2 TLB has 512 entries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLB misses handled in hardware</td>
</tr>
</tbody>
</table>
Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories 😞
  - Caching gives this illusion 😊
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors
Virtual Machines
Virtual Machines

- Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources

- Virtualization has some performance impact
  - Feasible with modern high-performance computers

- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC
Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest
Virtual Machine Monitor (Hypervisor)

Diagram showing layers of the system:

- (a) Processes
- Programming interface
- Kernel
- Hardware

(b) Processes
- Kernel
- VM1
- VM2
- VM3
- Virtual-machine implementation
- Hardware
Example: Timer Virtualization

- In native machine, on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process

- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM

- If a VM requires timer interrupts
  - VMM emulates a virtual timer
  - Emulates interrupt for VM when physical timer interrupt occurs
Instruction Set Support

- User and System modes
- Privileged instructions only available in system mode
  - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
  - Current ISAs (e.g., x86) adapting