Ideal Memory

- Infinitely large and very short access time
- However, this is technologically infeasible
- Solution: Memory hierarchy
  - Large and slow units, and
  - Small and fast units
- Levels in hierarchy characterized by
  - Total capacity
  - Access time
  - Transfer rate - bandwidth
  - Unit of transfer
  - Cost per byte

Big Memory
- Longer wires
- Higher Fan out

CPU

Small Memory
Levels of the Memory Hierarchy

Upper Level

- CPU Registers
  - 100s Bytes
  - <1 ns
  - $10/MMByte

- Cache
  - 10s-1000s K Bytes
  - 1-10 ns
  - $10/MMByte

- Main Memory
  - 10s-1000s M Bytes
  - 60ns-200ns
  - $1/MMByte

- Disk
  - 100s-1000s G Bytes
  - 10 ms (10,000,000 ns)
  - $0.0031/MMByte

Lower Level

- Prog./compiler
  - 4-8 bytes
  - HW cache control
  - 16-128 bytes
  - OS
  - 512-4K bytes

Power 7 On-Chip Caches

- 32KB L1 I$/core
- 32KB L1 D$/core
- 3-cycle latency

- 256KB Unified L2$/core
- 8-cycle latency

- 32MB Shared L3$
- Embedded DRAM
- 25-cycle latency
Major Properties

♦ Inclusion Property - \( M_i \subseteq M_{i+1} \)
  • \( M_i \) - upper level, closer to the CPU

♦ Coherence property - copies in successive memory levels must be consistent

♦ Two update methods
  (i) Write-through - immediate update
  (ii) Write-back - delay update of \( M_{i+1} \) until the item in \( M_i \) is replaced.

Replacement policies

The Principle of Locality

♦ The Principle of Locality:
  • Program accesses a relatively small portion of the address space at any instant of time.

♦ Two Different Types of Locality:
  • Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  • Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
Memory Hierarchy Technology

♦ Random Access (RAM): same access time for all locations
  • DRAM: Dynamic Random Access Memory
    » High density, low power, cheap, but slow
    » Dynamic: need to be “refreshed” regularly
  • SRAM: Static Random Access Memory
    » Low density, high power, expensive, fast
    » Static: content will last “forever” (until lose power)
  • The Main Memory: DRAMs; Caches: SRAMs

♦ “Non-so-random” Access Technology:
  • Access time varies from location to location and from time to time
  • Examples: Disk, CDROM

Layout of SRAM cell vs DRAM cell
Memory Hierarchy - General Principles

- Solve the problems of
  - Speed gap
  - Memory size
- Cache - Main memory: Speed Block
- Main memory - Disk: Capacity Page
- A computer system may have one of these or both

Processor - Memory Gap (latency)

Four-issue 3GHz superscalar accessing 100ns DRAM could execute 1,200 instructions during time for one memory access!
Memory Hierarchy: Terminology

- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit_Rate**: the fraction of memory access found in the upper level
- **Miss**: data needs to be retrieved from a block in the lower level (Block Y)
  - **Miss_Rate** = 1 - (Hit_Rate)

Average Memory Access time (AMAT)

- **Simplified expression**
  \[ \text{AMAT} = \text{Hit}_\text{Rate} \times T_{\text{upper}} + (1 - \text{Hit}_\text{Rate}) \times T_{\text{lower}} \]
- **A more precise expression**
  \[ \text{AMAT} = \text{Hit}_\text{Rate} \times \text{Hit}_\text{Time} + \text{Miss}_\text{Rate} \times \text{Miss}_\text{Penalty} \]
  \[ \text{Hit}_\text{time} = T_{\text{upper}} + \text{Time} \_ \text{to} \_ \text{determine} \_ \text{hit/miss} \]
  \[ \text{Miss}_\text{penalty} = \text{Time} \_ \text{to} \_ \text{determine} \_ \text{hit/miss} + \text{Time} \_ \text{to} \_ \text{replace} \_ \text{a} \_ \text{block} \_ \text{in} \_ \text{the} \_ \text{upper} \_ \text{level} + \text{Time} \_ \text{to} \_ \text{deliver} \_ \text{the} \_ \text{data} \_ \text{to} \_ \text{processor} \]
- **Hit_Time << Miss_Penalty** (50 - 100 cycles upon miss in cache; tens of thousands cycles upon miss in main memory)
Block (Page) Size

\[ \text{AMAT} = \text{Hit Rate} \times \text{Hit Time} + \text{Miss Rate} \times \text{Miss Penalty} \]

Impact on Processor Performance

♦ Given a processor with
  • 1 cycle on-chip cache
  • Base CPI = 1.1
  • 50% arith/logic, 30% ld/st, 20% control
♦ Suppose that 10% of memory data operations get 50 cycle miss penalty
♦ Suppose that 1% of instructions fetch get same miss penalty
♦ CPI = Base CPI + average stalls per instruction
  = 1.1(cycles/ins) + [0.30 (DataMops/ins) x 0.10 (miss/DataMop) x 50 (cycle/miss)] + [
    1 (InstMop/ins) x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1
♦ 64% of the time the proc is stalled waiting for memory!
♦ AvgMemAccessTime=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=
  2.54
Four Questions for Memory Hierarchy

♦ Q1: Where can a block be placed in the upper level?  

(Block placement)

♦ Q2: How is a block found if it is in the upper level?  

(Block identification)

♦ Q3: Which block should be replaced on a miss?  

(Block replacement)

♦ Q4: What happens on a write?  

(Write strategy)
Virtual Memory

Main memory - secondary storage (hard drive) hierarchy:
Objective: Provide a very large memory space
Example: Virtual memory space \(2^{32} = 4 \text{ GByte}\)
Physical memory \(2^{27} = 128 \text{ Mbyte}\)

virtual and physical address space partitioned into parts of equal size: \(2 \text{ KByte/page} = 2^{11}\)

Address Mapping

Processor \[\text{VA} \rightarrow \text{Main Memory} \rightarrow \text{Disk}\]

Address Trans Mechanism

physical address \[\text{PA} \rightarrow \text{Main Memory} \rightarrow \text{Secondary Memory}\]

Disk

OS performs this transfer

Page 9
Paging Organization (Protection)

Virtual Address (VPN) offset

Kernel/User Mode
Read/Write

Protection Check
Address Translation

Exception?

Virtual Page No. (VPN) offset

Page Table

index into page table

Page Table Base Addr

V Access Rights Frame no.

Physical Address Physical Page No. (PPN) offset

Physical memory address

Physical memory address

Page Tables in Physical Memory

User 1 Virtual Address Space

User 2 Virtual Address Space

Physical Memory

VA1

PT

User

1

VA1

PT

User

2

VA1

PT

User

1

VA1

PT

User

2

VA1

PT

User

1

VA1
Page Table

- Page Table Entry (PTE) contains:
  - A bit to indicate if a page exists (V)
  - PPN (physical page number) for a memory-resident page
  - DPN (address on disk) for a page on the disk
  - Access rights bits for protection
- OS sets the Page Table Base Register whenever active user process changes

Address Translation Overhead

- Page table is a large data structure in memory: entries!
- Two memory accesses for every load, store, or instr. fetch!!!
- “Cache” the address translations - use content-addressable memory (CAM, a.k.a Associative memory)
Translation Lookaside Buffer (TLB)

Rely on temporal locality:
keep only the most recently used pages in TLB
32 to 512 entries
TLB_Hit_time = 1 cycle
Miss_penalty = 20-100 cycles
Hit_rate = 99%

X cycles to access memory

\[
\text{Memory\_Hit\_time} = 0.99(X+1) + 0.01(2X) = 1.01X + 1
\]

= 41.4 for \(X=40\)

I-TLB & D-TLB

Page Table Size Problem

♦ Increase page size
  • Also: transfer of pages to/from disk more efficient

♦ However:
  • Lower Hit_rate
  • Internal fragmentation

♦ Solution 1: Hashing (reduce from \(2^{21}\) to \(2^{16}\))
  • Hashing function takes 21 bits and outputs 16 bits. It performs some arithmetic operation (ignoring overflows) and then a mod \(k\) operation (\(k=2^{16}\)). The result is an integer in \([0,k-1]\)
  • The resulting table (called inverted page table) of 64K entries is sometimes divided into several smaller tables
  • Possible aliasing
    » The table entry must include the virtual page number

Solution 2: Segmented Memory

Paged segments
* No table has more than 2048 entries
* Only active PTs reside in memory
* Penalty?


Q3: Page Replacement Policies

♦ Goal: minimize number of page faults
♦ Effectiveness depends on:
  • (1) Page size (2) No. of available frames (3) Program behavior
♦ Policies - examples:
  • Optimal algorithm (upper bound)
  • Random replacement (lower bound)
  • First-In First-Out (FIFO) - longest time, e.g., 12131415
  • Least Recently Used (LRU) - was not used recently for the longest time
  • Least Frequently Used (LFU) - least referenced
  • Example: Page x 1111 0000
      Page y 0000 1000
      Page z 0000 0111
### LRU vs. FIFO

**page size:** 4 words; 3 page frames: a, b, c; 10 pages: 0, 1, 2, ..., 9

**Word Trace:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 28, 29, 30, 8, 9, 10, 4, 5, 12, 4, 5

**Page Trace:** 0 1 2 4 2 3 7 2 1 3 1

<table>
<thead>
<tr>
<th>PF</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>7</th>
<th>2</th>
<th>1</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Faults</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>OPT</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Faults</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>FIFO</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Faults</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

**Hit Rate**

### Implementing FIFO & LRU

- **FIFO** - linked list
  - Add new page at tail, remove from head
  - Improvement: Put pages on the free-page list, scan it upon page fault and re-establish (soft page fault - no disk access)

- **LRU** - most commonly used policy
  - **Use** (or **Reference**) bit - set when page accessed; OS periodically sorts and moves the referenced pages to the top & resets all Use bits
  - A more accurate implementation - (software) **Age Counter**, +1 if not referenced, reset if referenced
  - Must provide timer interrupt to update LRU bits
The four questions (virtual storage)

♦ 1) Where can a page be placed? Fully associative
♦ 2) How is a page found? TLB + PTs
  • Page tables map virtual address to physical address
  • TLBs make virtual memory practical - temporal and spatial
    locality
♦ 3) What block is replaced on miss? Replacement
    policy
♦ 4) How are writes handled? Always Write Back
  • Minimize penalty by using a Dirty Bit

TLB Summary

Translation Lookaside Buffer or TLB

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>Dirty</th>
<th>Ref</th>
<th>Valid</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Access bits: No_access, Read_only, Execute_only, RW&Execute
AMD’s Bobcat core