Big Memory

CPU

Small Memory

• Longer wires
• Higher Fan out

Ideal Memory

♦ Infinitely large and very short access time
♦ However
  • Very expensive
  • 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
Levels of the Memory Hierarchy

- **CPU Registers**: 100s Bytes, <1 ns, $10/\text{MByte}
- **Cache**: 10s-100s K Bytes, 1-10 ns, $1/\text{MByte}
- **Main Memory**: 10s-100s M Bytes, 60ns-200ns, $1/\text{MByte}
- **Disk**: 100s G Bytes, 10 ms, (10,000,000 ns), $0.0031/\text{MByte}

Power 7 On-Chip Caches

- 32KB L1 I$/\text{core}
- 32KB L1 D$/\text{core}
- 3-cycle latency
- 256KB Unified L2$/\text{core}
- 8-cycle latency
- 32MB Shared L3$
- 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**

```
Lower Level
Upper Level
To Processor
Copy of X
From Processor
```

**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: 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

♦ Sequential Access Technology: access time linear in location (e.g., Tape)

1-T DRAM Cell

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**
  \[
  \text{AMAT} = \text{Hit Rate} \times T_{upper} + (1 - \text{Hit Rate}) \times T_{lower}
  \]
- **A more precise expression**
  \[
  \text{AMAT} = \text{Hit Rate} \times \text{Hit Time} + \text{Miss Rate} \times \text{Miss Penalty}
  \]
  \[
  \begin{align*}
  \text{Hit time} & = T_{upper} + \text{Time to determine hit/miss} \\
  \text{Miss penalty} & = \text{Time to determine hit/miss} + \text{Time to replace a block in the upper level} + \text{Time to deliver the data to processor}
  \end{align*}
  \]
- **Hit Time << Miss Penalty** (50-100 cycles upon miss in cache; tens of thousands cycles upon miss in main memory)
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)] × 0.10 (miss/DataMop) × 50 (cycle/miss) + 
    [ 1 (InstMop/ins) × 0.01 (miss/InstMop) × 50 (cycle/miss)]
  = (1.1 + 1.5 + 0.5) cycle/ins = 3.1
- 64% of the time the proc is stalled waiting for memory!

\[
\text{AvgMemAccessTime} = \frac{1}{1.3} \times [1+0.01 \times 50] + (0.3/1.3) \times [1+0.1 \times 50] = 2.54
\]
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 - disk hierarchy:
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
\[ \text{VA} \]
Addr Trans Mechanism
\[ \text{PA} \]
physical address
Main Memory
Secondary Memory
Disk

OS performs this transfer

missing page fault

fault handler

\text{VA} \rightarrow \text{PA} \rightarrow \text{Main Memory} \rightarrow \text{Secondary Memory} \rightarrow \text{Disk}

Hit_time = 50-100 CPU cycles
Miss_penalty = \(10^6\) cycles
Miss_rate = 1%
Paging Organization (Protection)

Virtual Address → Virtual Page No. (VPN) → offset

VA

Kernel/User Mode

Read/Write

Protection Check

Address Translation

Exception?

Physical Address → Physical Page No. (PPN) → offset

VA → page no., offset

Page Table

Page Table Base Addr

index into page table

Page Table

V Access Rights

Frame no.

table located in physical memory

physical memory address

PA

Page Tables in Physical Memory

User 1 Virtual Address Space

User 2 Virtual Address Space

PT

User

PT

User

Physical Memory
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)

Virtual memory $2^{32} = 4$ GByte
Physical memory $2^{27} = 128$ MByte
Page size = 2 KByte

No. of entries =
If tag & Page # match
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
Memory_Hit_time = 0.99 \( (X+1) \) + 0.01 \( (2X) \)
= \( 1.01X + 1 \)
= 41.4 for \( X=40 \)

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
  - 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, 16,17, 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 |
|---|---|---|---|---|---|---|---|---|---|---|
| PF | 0 | 1 | 2 | 4 | 2 | 3 | 7 | 2 | 1 | 3 |
| a | 0 | 0 | 0 | 4 | 4 | 4 | 7 | 7 | 7 | 3 |
| b | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 1 | 1 | 1 |
| c | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Faults | * | * | * | * | * | * | * | * | * | * |

| OPT |
|---|---|---|---|---|---|---|---|---|---|---|
| a | 0 | 0 | 0 | 4 | 4 | 4 | 7 | 7 | 7 | 3 |
| b | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| c | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Faults | * | * | * | * | * | * | * | * | * | * |

| FIFO |
|---|---|---|---|---|---|---|---|---|---|---|
| a | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 2 | 2 | 2 |
| b | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 1 |
| c | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 | 3 | 3 |
| Faults | * | * | * | * | * | * | * | * | * | * |

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>
</table>

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

Core Floor Plan

Floating Point Unit

Data L2 TLB

Bus Unit

L2 Sub Array

L2 TAG

Instruction Cache

Int TLB/Tag

Branch Predict

Code ROM

RCB

Data Cache

Data Tag/TLB

Load Store Unit

Source: AMD 2010