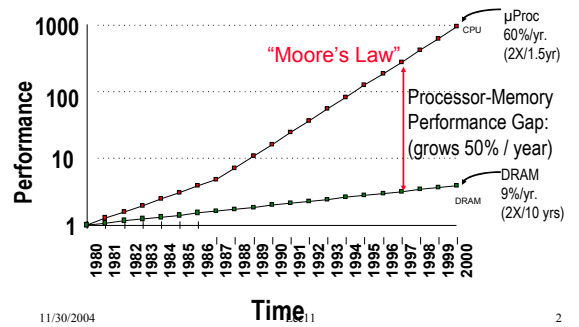


Lecture 11 Memory Hierarchy

Instructor: Jun Yang

Motivation

Processor-DRAM Memory Gap (latency)

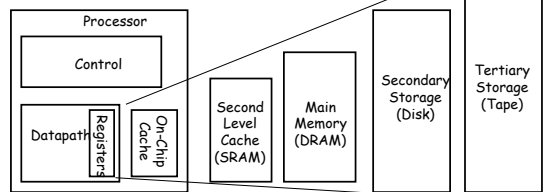


The Goal: Illusion of Large, Fast, Cheap Memory

- Goal: a large and fast memory
- Fact:
 - Large memories are slow
 - Fast memories are small
- How do we create a memory that is large, cheap and fast (most of the time)?
 - Hierarchy

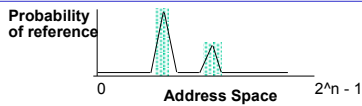
Memory Hierarchy of a Modern Computer System

- By taking advantage of the principle of locality:
 - Present the user with as much memory as is available in the cheapest technology.
 - Provide access at the speed offered by the fastest technology.

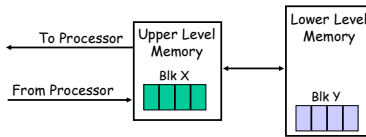


Speed (ns)	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes)	100s	Ks	Ms	Gs	Ts

Memory Hierarchy: Why Does it Work? Locality!



- **Temporal Locality** (Locality in Time):
=> Keep most recently accessed data items closer to the processor
- **Spatial Locality** (Locality in Space):
=> Move blocks consists of contiguous words to the upper levels



11/30/2004

Lec11

5

Memory Hierarchy Technology

- **Random Access:**
 - "Random" is good: access time is the same for all locations
 - **DRAM:** Dynamic Random Access Memory
 - ❖ High density, low power, cheap, 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)
- **"Not-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)
- **We will concentrate on random access technology**
 - The Main Memory: DRAMs + Caches: SRAMs

11/30/2004

Lec11

6

Introduction to Caches

- **Cache**
 - is a small very fast memory (SRAM, expensive)
 - contains copies of the most recently accessed memory locations (data and instructions): **temporal locality**
 - is fully managed by hardware (unlike virtual memory)
 - storage is organized in **blocks** of contiguous memory locations: **spatial locality**
 - unit of transfer to/from main memory (or L2) is the cache block
- **General structure**
 - n blocks per cache organized in s sets
 - b bytes per block
 - total cache size $n \cdot b$ bytes.

11/30/2004

Lec11

7

Caches

- **For each block:**
 - an address tag: unique identifier
 - state bits:
 - ❖ (in)valid
 - ❖ modified
 - the data: b bytes
- **Basic cache operation**
 - every memory access is first presented to the cache
 - **hit:** the word being accessed is in the cache, it is returned to the cpu
 - **miss:** the word is not in the cache,
 - ❖ a whole block is fetched from memory (L2)
 - ❖ an "old" block is evicted from the cache (kicked out), which one?
 - ❖ the new block is stored in the cache
 - ❖ the requested word is sent to the cpu

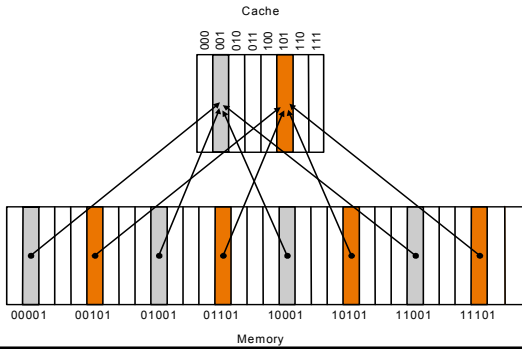
11/30/2004

Lec11

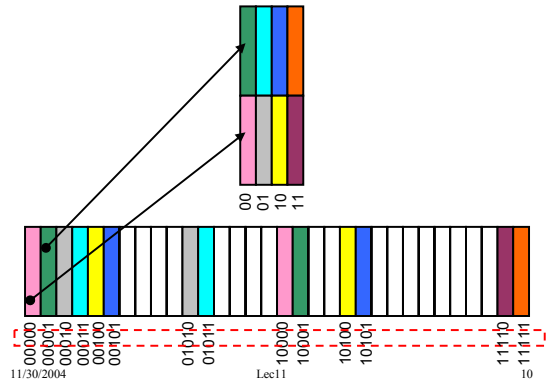
8

Direct Mapped Cache

- Cache stores a subset of memory blocks
- Mapping: address is modulo the number of blocks in the cache

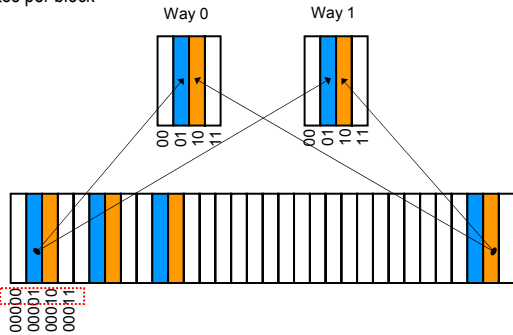


Block Size > 4 bytes (1 word)



Two way set-associative mapping

4 bytes per block



11/30/2004

Lec11

11

Addressing the Cache

- Direct mapped cache: one block per set.



- Set-associative mapping: n/s blocks per set.



- Fully associative mapping: one set per cache ($s = n$).



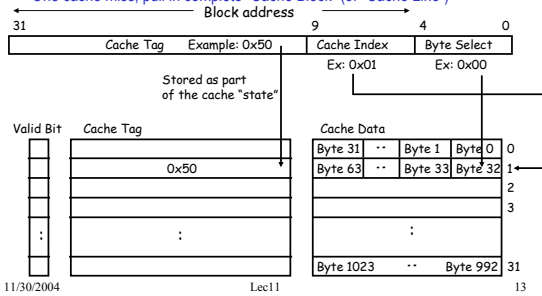
11/30/2004

Lec11

12

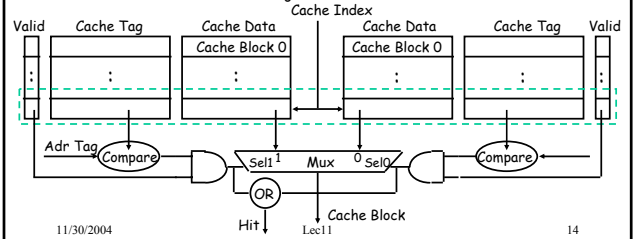
Example: 1 KB Direct Mapped Cache with 32 B Blocks

- For a $2^{**} N$ byte cache:
 - The uppermost (32 - N) bits are always the Cache Tag
 - The lowest M bits are the Byte Select (Block Size = 2^M)
 - One cache miss, pull in complete "Cache Block" (or "Cache Line")



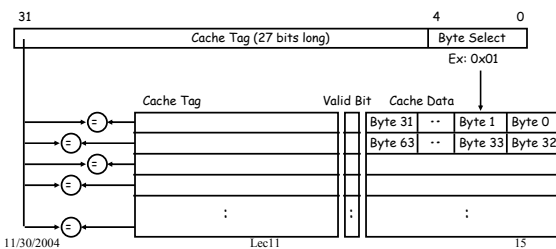
Set Associative Cache Architecture

- N-way set associative:** N entries for each Cache Index
 - N direct mapped caches operates in parallel
- Example: Two-way set associative cache**
 - Cache Index selects a "set" from the cache
 - The two tags in the set are compared to the input in parallel
 - Data is selected based on the tag result



Example: Fully Associative Architecture

- Fully Associative Cache**
 - Forget about the Cache Index
 - Compare the Cache Tags of all cache entries in parallel
 - Example: Block Size = 32 B blocks, we need N 27-bit comparators
- By definition: Conflict Miss = 0 for a fully associative cache



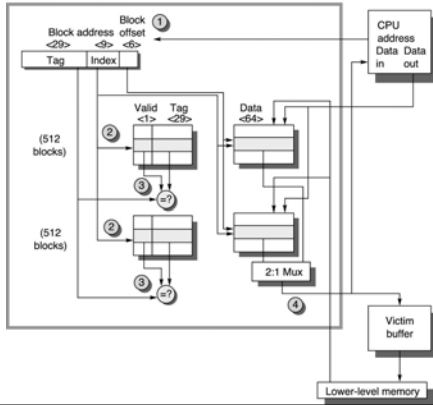
Cache Organization - Example

- Instruction & Data caches:**
 - Because instructions have much lower miss rates (see Fig 5.8): split the cache (at L1 level) into instruction and data. Otherwise, unified.
 - Main advantages: no interference, data misses do not stall instruction fetch, optimize for different access patterns etc.

DEC Alpha 21264 Cache

- 64K I & D caches (L1), 2-way set associative;
- b = 64 bytes;
- write-back; write allocate on write miss.
- Organization shown in Fig. 5.7: data RAM is 64 bytes wide (no multiplexer), 3 bits of offset are used (with index) to select a 64 bit word in the block.
- Read & write hits: 3 cycles, pipelined.
- Victim buffer: 8 entries.

Alpha 21264 Cache Organization



Block Replacement

- Block replacement
 - not an issue with direct mapped
 - replacement strategy is more important in small caches than in large ones (see Fig. 5.6).
 - replacement policies:
 - ❖ LRU: has been unused for the longest time. good temporal locality, complex state
 - ❖ pseudo-random: randomly selected
 - ❖ FIFO: oldest block. Difference with LRU?

11/30/2004

Lec11

18

Cache Design Tradeoffs

- Design Parameters
 - cache size
 - block size
 - associativity
- Cache Size
 - bigger: exploits temporal locality better
 - but is slower: slower cycle time
- Associativity
 - lower: faster cycle time
 - higher: lower miss rate
 - Rule 1: miss rate of 8-way s-a \approx fully s-a.
 - Rule 2: miss rate of d-m of size N \approx that of a 2-way s-a of size N/2.
 - Hit time in d-m is shorter than in s-a.
- Block Size
 - spatial locality within a block
 - too small: too many transfers of data from memory
 - too large: wasted bandwidth, unused transferred data
 - large block size reduces compulsory misses, taking advantage of spatial locality;
 - not always good: increased miss penalty
 - tradeoff: latency, bandwidth and block size.

11/30/2004

Lec11

19

Write Policies

- Writes are hard
 - read: concurrently check tag and read data
 - write is destructive, so it must be slower
- Trade-offs
 - Write-back:
 - ❖ uses less bus bandwidth,
 - Write-through:
 - ❖ keeps MM consistent with CM,
 - ❖ good for DMA.
- Write strategies
 - when to update memory?
 - ❖ on every write (*write-through*)
 - ❖ when a modified block is replaced (*write-back*)
 - what to do on a *write miss*?
 - ❖ fetch the block to cache (*write allocate*), used with write-back
 - ❖ do not fetch (*write around*), used with write-through

11/30/2004

Lec11

20

Write Buffers

- Write Buffers (for wrt-through)
 - buffers words to be written in L2 cache/memory along with their addresses.
 - 2 to 4 entries deep
 - all read misses are checked against pending writes for dependencies (associatively)
 - allows reads to proceed ahead of writes
 - can coalesce writes to same address
- Write-back Buffers
 - between a write-back cache and L2 or MM
 - algorithm
 - ❖ move dirty block to write-back buffer
 - ❖ read new block
 - ❖ write dirty block in L2 or MM
 - can be associated with victim cache (later)

