

Memory System Design

Memory - What Is It?

- Memory:
 - Devices used to store instructions and data required for program execution
 - Examples: RAM, disks, tapes, ...
- Characteristics:
 - **Cost**: generally defined i.t.o. dollars/byte
 - **Access time**
 - **Access modes**: sequential vs. random access
 - **Alter-ability**: read-only memory (ROM) vs. read-write memory

Random Access Memories

- Semiconductor memory - volatile
- Static vs. dynamic RAM
- Storage cells are organized as rectangular array
 - 1-D addressing scheme
 - 2-D addressing scheme
- An n -bit memory address can refer to 2^n words of memory

Memory System Design: Goals

The main goals of memory design are to have:

- As large a memory as possible
 - programs expand to fill all available memory
 - large memories allow more users, larger problems, etc...
- As fast a memory as possible
 - the memory system should be able to keep up with the processor (instructions and data references)
 - the memory system should allow for concurrent I/O operations as well
- And, as low a cost as possible
- Problem: conflicting goals!

Basic Principles

- **Principle of locality**: memory locations are not accessed with uniform frequency; certain locations get accessed a lot more often than others

Less than ten percent of the instructions of a program account for more than ninety percent of its run time

- **Spatial locality principle**: if a location is accessed, then nearby locations are likely to be accessed in the nearby future
- **Temporal locality principle**: if a location is accessed, then it is likely to be accessed again in the nearby future

The principle of locality enables the design of cost effective memory systems

Memory Hierarchy

- Memory is organized in levels
- Highest level (closest to the processor) is the fastest, it is the most expensive (per bit) because of its speed, it is also the smallest (otherwise it will be too costly)
- All levels are organized in blocks, when a data item (word) is transferred between levels, the whole block containing that item is transferred in an attempt to exploit spatial locality
- Lowest level is the cheapest (per bit) and it is the largest of the levels
- A well designed memory hierarchy will deliver a performance level close to that of its fastest level at a cost (per bit) of its cheapest (and largest) level

A Common Memory Hierarchy Example

- CPU \Rightarrow Cache \Rightarrow Main memory
- **Cache:** a small, transparent, fast, memory
 - cache retains copies of (blocks containing) recent references
 - if data is found in the cache, it is delivered to the CPU
 - otherwise data is brought from main memory which is usually the second level of the hierarchy
- **Cache is usually completely transparent to the application programmer**
- Memory hierarchy organization
 - The main memory is logically divided into blocks
 - Cache is organized as a smaller set of frames of same size as main memory blocks
 - A cache frame is either empty or it is occupied by a copy of a main memory block
 - The cache contains a directory of the main memory blocks currently in the cache

Basic Operation of a Memory Hierarchy

- When a data item is referenced:
 - the main memory block number is extracted from the address
 - the block number is looked up in the cache directory
 - if a copy of the block is currently in the cache the data is extracted from the appropriate cache frame: we have a hit
 - otherwise: we have a miss, a copy of the block is transferred from the main memory to one cache frame (1 frame houses 1 block)
- If the block frame chosen to receive the missing block is not empty when a miss occurs, then its contents must be ejected
 - if the contents have not been modified, they can be disregarded
 - otherwise, they must be copied back to the main memory

Terminology

- **Hit**: when a data item is found at a level
- **Miss**: if a reference is not satisfied at a memory level and must be forwarded to a lower level
 - **Hit ratio or hit rate**: the ratio of hits to the total number of references to a particular memory level
 - **Miss ratio**: $1 - \text{hit ratio}$
 - **Hit time**: time to retrieve a data item from a memory level on a hit
 - **Miss time**: (assume a hit in the next level of the hierarchy)
 - * $\text{Miss time} = \text{hit time} + \text{miss penalty}$
 - * $\text{Miss penalty} \leq \text{Access time} + \text{transfer time}$
- For two level memory hierarchy
 - Av. mem-access time = $\text{Hit-time} * \text{hit-rate} + \text{Miss-time} * \text{miss-rate}$
 - = $\text{Hit-time} + \text{Miss-penalty} * \text{miss-rate}$

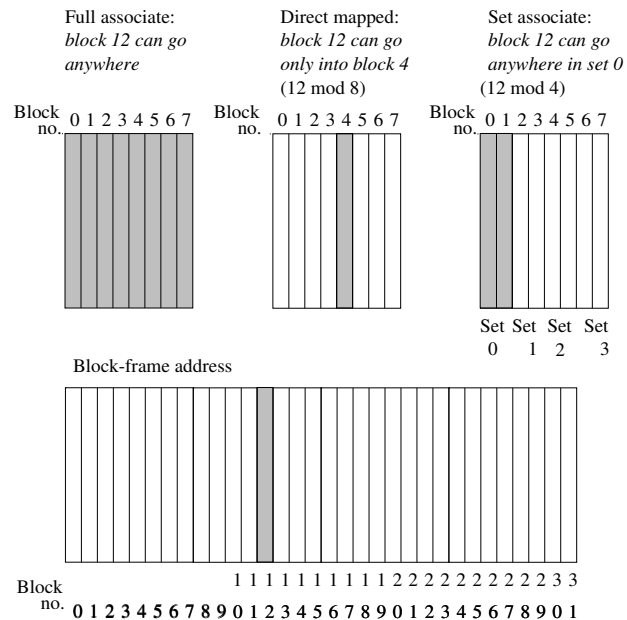
Memory Hierarchy Design Issues

- Implications of Hierarchy on CPU Design
 - CPU must allow for variable memory access time
 - CPU must allow for interrupts related to memory hierarchy
- Design Questions
 - **Block placement/identification**: where is a block placed when it is copied to a higher level in the hierarchy?
 - **Block replacement**: which block should be ejected from a given level to make room for a missing block?
 - **Write strategy**: if a block copy is modified in a higher level, when are the lower levels updated? (data inconsistency problem)

Placement Policy

This policy determines which frame of the cache should receive the copy of a given block; there are three major placement policies

- Direct mapped
- Full associative
- Set associative

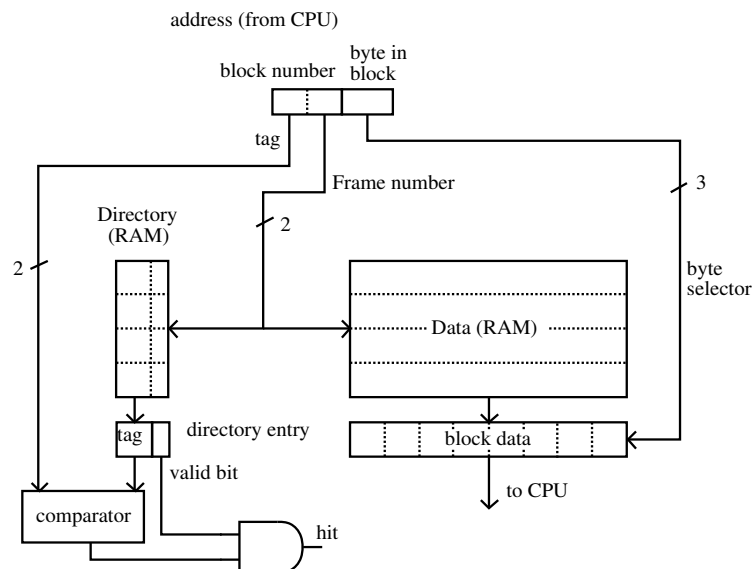


Direct Mapped Caches

- The direct mapped policy assigns block number i to frame number $(i \bmod f)$ where f (a power of 2) is the number of frames in the cache
 - Simple but inflexible - may lead to thrashing.
 - Implementation: The cache directory contains an entry for each frame, consisting of
 - * a valid bit indicating whether or not the entry contains a block
 - * the tag of the block occupying the frame (if valid)
- Lookup: The least significant m bits of the block number are used to index the cache directory
 - If the indexed cache directory entry has an invalid bit, or if the tag in the indexed entry does not match the tag of the address, we have a miss
 - Otherwise, we have a hit
- Replacement policy: trivial!

Direct Mapped Cache: Example

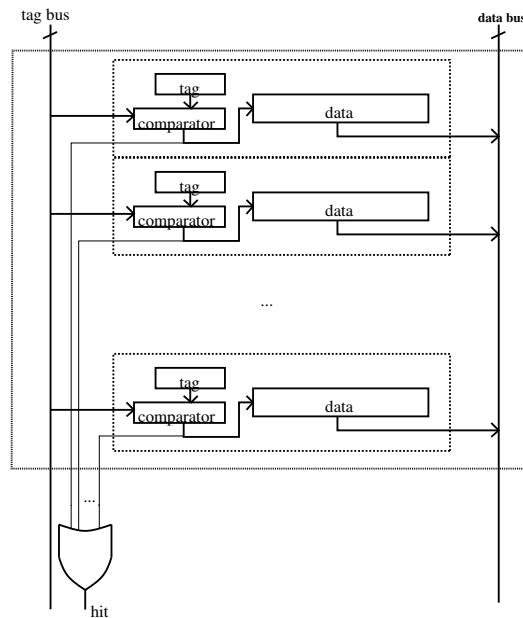
Block size = 8 bytes, main memory size = 16 blocks, cache size = 4 frames



Fully Associative Caches

- In fully associative caches, a main memory block may occupy any frame of the cache
 - Lookup is no longer as simple as direct mapped caches: we must search all the directory entries for a given tag
 - The search is performed by the hardware using an associative memory
- Associative memory: a memory consisting of pairs of items
 - a tag
 - and the associated data
 - If a tag is presented to the associative memory, then the memory will produce the data associated with the tag that matches the given tag (if any)

Associative Memory



Placement & Replacement

The placement policy determines where a block copy would go in the cache

- Random placement
- LRU (least recently used) placement:
 - idea:
 - if a frame has not been used for a while, it is unlikely to be used soon
- For small caches, LRU is better but expensive: usually an approximation of LRU is used
- For large enough caches, little performance difference

Set Associative Caches

Compromise between direct mapped associative caches

- Cache is organized as a set of fully associative caches
- Each block maps to exactly one set (similar to direct mapped)
- Within the set the block can occupy any frame (similar to fully associative)
- Achieves good hit ratio at reasonable hardware cost
- Example: The VAX-11/780 cache
 - Two-way set associative (set size = 2)
 - Block size = 8 bytes; Cache size = 1024 blocks
 - Random replacement; Write through policy; No write allocate

Write Policy

- If data is already in cache
 - **Write through**: main memory updated every time the cache is modified
 - * use write buffers to reduce stalls
 - **Write back**: data is written to the cache only; the main memory is updated when the contents of a “dirty” frame are ejected
- On a write miss we have two options
 - **Write allocate** (also called fetch on write): the block is first fetched into the cache and modified there, this is typically used with the write-back scheme
 - **No write allocate** (also called write around): the data is written directly into main memory; the affected block is not fetched

Cache Performance

- CPU time = (CPU execution clock cycles + memory stall clock cycles)
× clock cycle time
- Memory stall clock cycles = (reads × read miss rate × read miss penalty
+ writes × write miss rate × write miss penalty)
- Memory stall clock cycles = memory accesses × miss rate × miss penalty

Cache Performance (Cont'd.)

- CPU time = IC × (CPI_{execution} + memory accesses per instruction ×
miss rate × miss penalty) × clock cycle time
- Misses per instruction = memory accesses per instruction × miss rate
- CPU time = IC × (CPI_{execution} + misses per instruction × miss
penalty) × clock cycle time

Improving Cache Performance

- Average memory-access time = hit time + miss rate \times miss penalty (ns or clocks)
- Improve performance by:
 - Reduce the miss rate
 - Reduce the miss penalty, or
 - Reduce the time to hit in the cache

Classifying Misses

- **Compulsory**: The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*. (Misses in infinite cache)
- **Capacity**: If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. (Misses in size \times cache)
- **Conflict**: If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*. (Misses in n-way associative size \times cache)

Techniques for Reducing Miss Rate

- Reduce misses via larger block size.
- Reduce misses via higher associativity.
 - Thumb rule: Miss rate for direct mapped cache of size $n \approx$ miss rate for fully associative cache of size $n/2$
 - Caveat: Execution time is the only final measure!
 - * Will clock cycle time increase?
- Reducing misses via victim cache
 - How to combine fast hit time of direct mapped, yet still avoid conflict misses?
 - Add buffer to place data discarded from cache.
 - Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4KB direct mapped data cache.

Techniques for Reducing Miss Rate (Cont'd)

- Reducing misses via Pseudo-Associativity
 - How to combine fast hit time of direct mapped and have the lower conflict misses of 2-way set associative cache?
 - Divide cache: on a miss, check other half of cache to see if it is there, if so, there is a pseudo-hit (slow hit).
- Reducing misses by hardware and software prefetching data.

Reducing Misses by Compiler Optimizations

- Instructions

- Reorder procedures in memory so as to reduce misses
- Profiling to look at conflicts

- Data

- **Merging Arrays**: Improve spatial locality by single array of compound elements vs. 2 arrays
- **Loop Interchange**: Change nesting of loops to access data in order stored in memory.
- **Loop Fusion**: Combine 2 independent loops that have same looping and some variables overlap.
- **Blocking**: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows.

Merging Arrays Example

```
/* Before */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val and key

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
      x[i][j] = 2 * x[i][j];
```

```
/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
      x[i][j] = 2 * x[i][j];
```

Sequential accesses instead of striding through memory every 100 words.

Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    d[i][j] = a[i][j] + c[i][j];
```

```
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {
    a[i][j] = 1/b[i][j] * c[i][j];
    d[i][j] = a[i][j] + c[i][j];
  }
```

2 misses per access to a and c vs. one miss per access.

Reducing Miss Penalty

Five techniques:

- Read priority over write on miss
- Sub-block placement
- Early Restart and Critical Word First on miss
- Non-blocking caches (Hit Under Miss)
- Second level cache

Miss Penalty Reduction: Second Level Cache

- L2 equations

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$$

$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$$

$$\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$$

- Definitions:

- *Local miss rate*: Misses in this cache divided by the total number of memory accesses to this cache (Miss Rate_{L2})
- *Global miss rate*: Misses in this cache divided by the total number of memory accesses generated by the CPU ($\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$)

Improving Hit Times

- Small and simple caches
- Avoiding address translation
- Pipelined writes
- Fast writes on misses via small sub-blocks