Chapter 3:
Caches

Lecturer: *Paul H J Kelly*

Autumn 2001

Department of Computing
Imperial College

Room 423
email: phjk@doc.ic.ac.uk

Course web site:
http://www.doc.ic.ac.uk/~phjk/AdvancedComputerArchitecture.html

CHAPTER 3

3.1 Memory price/performance

3.2 Memory Hierarchies

3.4 Direct-Mapped Caches

3.5 Block Placement

3.6 Block Replacement

3.7 Write Policies

3.8 Trace-Based Cache Analysis

3.9 Average memory access time

3.1 Memory price/performance

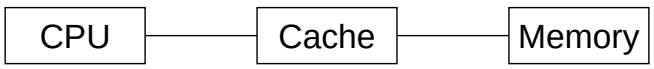
- Dynamic RAM (DRAM):
  - Square array, one transistor per bit, data stored as charge on capacitor
  - Very low power
  - Discharge to read, so must write back — so cycle time > access time
  - Charge slowly decays so needs periodic refresh
  - Address passed in two halves:
    - Row address (RAS) transfers data from a whole row into an on-chip buffer
    - Column address (CAS) selects required bit from buffer
    - Can issue one RAS then use many different CAS's to access the row buffer
  - Access time: ca.50ns (10-50 cycles)
  - Cycle time: ca.90ns
  - Commonly 64Mbit/chip or more
  - Various higher-performance interfaces available (SDRAM, RamBus, VDRAM)

Memory price/perf - cont'd

- Static RAM (SRAM)
  - 4–6 transistors/bit, VLSI die area 4–6× larger
  - Uses power just to hold data
  - 4–8× (or more) faster
  - 4–8× (or more) more expensive

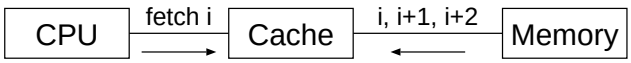
(H&P pp.9,41,429,437)

3.2 Memory Hierarchies



- Use a small amount of fast, expensive memory, backed up by a large amount of slower, cheaper memory which is large enough to store all data
- If an instruction refers to a datum which is not in the cache, it is automatically copied from the slow memory into the cache
- If most accesses can be satisfied by the cache, the average latency of memory accesses will be close to the cycle time of the cache
- We aim to achieve the performance of SRAM (expensive, fast) by using DRAM (cheaper, slower)

Why Might it Work?



- Loops: a small set of instructions is repeatedly executed. Instructions that were accessed in the recent past will hopefully still be in the cache when they are next required
- Procedure call stacks, arrays...

Temporal locality: recently referenced items are likely to be referenced again soon, e.g. loops, stacks, temporary variables

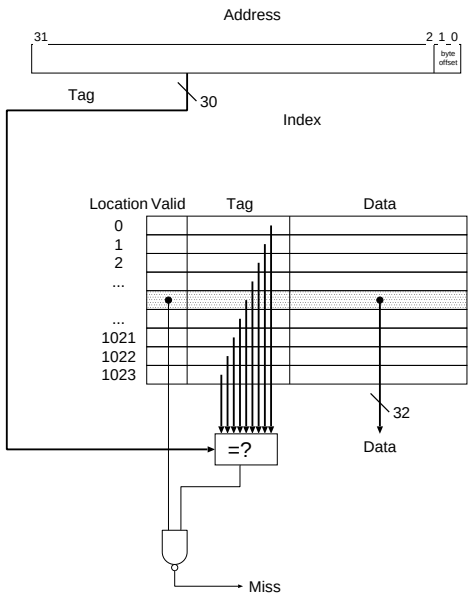
Spatial locality: programs tend to access items whose addresses are near one another, e.g. arrays, code segments, stacks

Hit ratio: the ratio of the number of cache hits to the number of cache accesses

Fundamental definitions

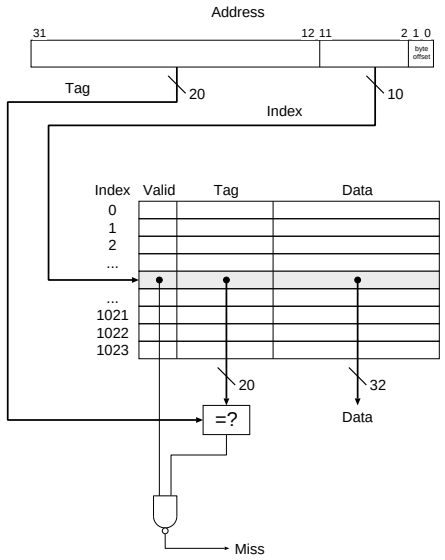
- **Block:** a $B \times N$-word cache provides fast storage for up to B blocks, each consisting of N words
- **Tag:** Each block held in the cache must be tagged with its main memory address
- **Read Hit:** If tag matches address, the data is supplied to the processor
- **Read Miss:** no match: a block must be fetched from memory
- **Allocate:** when a new block is fetched, it is stored in the cache
- **Replacement:** to make space for a new block to be allocated in the cache, it may be necessary to displace an existing block.

3.3 Fully-associative caches



- In a fully associative cache, each main memory block can be allocated to any location in the cache
- To check whether a given block is in the cache, must compare address with every tag

3.4 Direct-Mapped Caches



- In a direct-mapped cache, each main memory block can be allocated to just one location in the cache
- To check whether a given block is in the cache, compare address with just the tag for its location in the cache

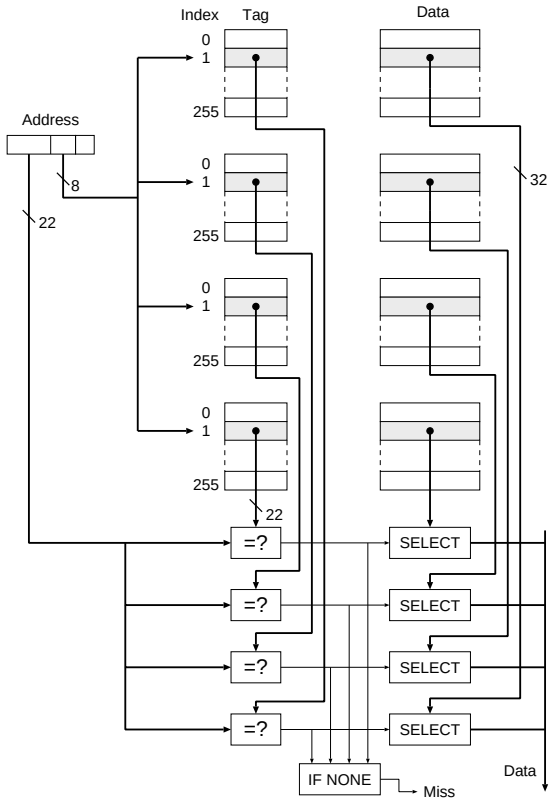
Cache Design Issues

- *Block placement:* where can a block be placed in the cache?
- *Block replacement:* which block should be replaced on a miss?
- *Write strategy:* what happens on a write?
- *Cache parameters:* e.g. block size, cache size, associativity

3.5 Block Placement

- *Direct-mapped:* block is limited to one location: only one comparator per cache
- *Set-associative:* block can be stored in any of n locations: n comparators required
- *Fully-associative:* block can be stored in any location

A 4-Way Set Associative Cache



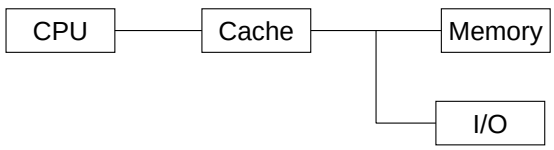
- *Fully-associative:* block can be stored in any location: each block needs a comparator
- *Direct-mapped:* block is limited to one location: only one comparator per cache
- Slower than direct-mapped (why?)
- Omitted valid bits and replacement stuff

3.6 Block Replacement

- On a cache miss, a block must be expelled before the new block is brought in from memory (unless the valid bit indicates that the cache block is empty)
- No choice for a direct-mapped cache
- Objective: to retain blocks which will be used again soon
  - *Optimal* (OPT): replace the block which will not be used for the longest time
  - *Random*
  - *Least-recently used* (LRU): replace the block which has been unused for the longest time
- LRU uses the past as an estimate of the future, i.e. the corollary of temporal locality: if recently used blocks are likely to be used again, the best candidate for replacement is the least recently used block

- **Q:** what is relationship between LRU and OPT?
- **Q:** how could LRU be implemented? Cost?
- Tradeoff between implementation cost and performance
- **Q:** how could LRU be approximated?
- See H&P pp.379 for comparison of LRU and random replacement
  - Random is nearly as good as LRU, especially for larger caches
  - Random is considerably easier to implement

3.7 Write Policies



Write-Through

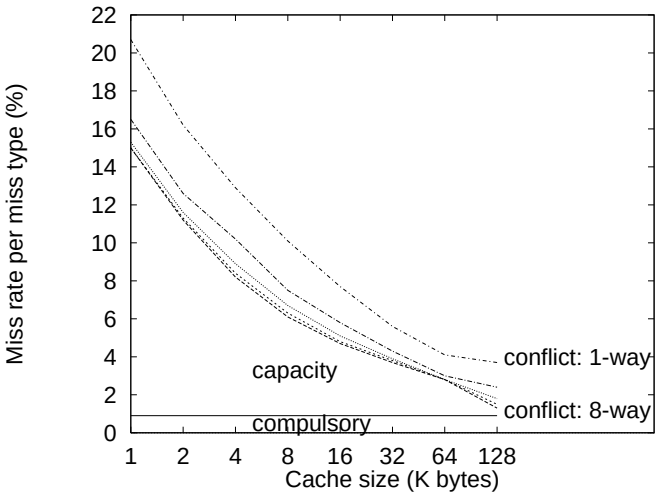
- Data is written to the block in the cache and the block in the lower-level memory
- Cycle time is that of memory
- Memory is always *coherent*

Write-Back

- Data is only written to the cache
- Modified blocks are written to memory only when replaced
- Cycle time of a hit is that of the cache
- Uses *dirty* bits to mark blocks which need to be written back when replaced

3.8 Trace-Based Cache Analysis

- Many design choices, e.g. degree of associativity, cache size, block size, replacement policy, write policy
- Most represent cost/performance tradeoffs. Cache simulators are used by designers to study these tradeoffs, and how cache parameters interact
- The effect of varying cache size and associativity is shown, based on a trace of a multiprogramming workload on a VAX with 32-byte blocks and LRU replacement (cf H&P pp 392)



Compulsory: the first access to a block requires it to be brought in from a higher level

Capacity: occurs when a new block is brought into a full cache

Conflict: occurs when more than one block maps to the same cache block: blocks must be discarded and later retrieved

(cf H&P pp.392)

Conclusions of Trace-Based Analysis

- Compulsory misses are constant
- 2:1 rule of thumb: miss rate of a direct-mapped cache of size $N \approx$ miss rate of a 2-way set-associative cache of size $N/2$
- Thrashing occurs when the cache is too small to hold the *working set* of the program: the memory system runs at the speed of memory

Trying it yourself

You can experiment with alternative cache designs yourself using the CSG Linux systems in two ways — instrumentation or simulation:

1: Hardware instrumentation

Compile your application as normal then use the 'hpm' command:

```
% cp ~phjk/ToyPrograms/C/MM.c ./
% gcc -O3 MM.c
% hpm -e PCL_L2CACHE_READWRITE,PCL_L2CACHE_MISS ./a.out
hpm: counter 0   : 21798432 PCL_L2CACHE_READWRITE
hpm: counter 1   : 15239794 PCL_L2CACHE_MISS
```

Ie. a miss rate of about 69%. Try also "`hpm -s`" and "`cat /proc/cpuinfo`".

Trying it yourself - simulation

2: Simulation, eg using Simplescalar (www.simplescalar.org)

Simplescalar is a parameterisable simulator with an instruction set based on MIPS. Compile your application using a special cross-compiler, then use one of the simulators:

```
% ~phjk/simplescalar/bin/sslittle-na-sstrix-gcc -O3 MM.c
% ~/simplescalar/simplesim-2.0/sim-cache ./a.out
...
ul2.miss_rate       0.4959 # miss rate (i.e., misses/ref)
...
```

The default (unified) level-2 cache is 1024 sets × 64 bytes × 4-way set associative. You can select other configurations using a command-line parameter...

The SimpleScalar toolkit comes with other simulators which allow you to study other aspects of processor design.

Specifying a cache configuration for SimpleScalar

The command-line parameter to control sim-cache's cache configuration has the following format:

```
<name>:<nsets>:<bsize>:<assoc>:<repl>

<name>   - name of the cache being defined
<nsets>  - number of sets in the cache
<bsize>  - block size of the cache
<assoc>  - associativity of the cache
<repl>   - block replacement strategy,
             'l'-LRU, 'f'-FIFO, 'r'-random
```

Cache levels can be unified by pointing a level of the instruction cache hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache configuration arguments. Most sensible combinations are supported, e.g.,

A unified l2 cache (il2 is pointed at dl2):

```
-cache:il1 il1:128:64:1:l -cache:il2 dl2
-cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
```

Or, a fully unified cache hierarchy (il1 pointed at dl1):

```
-cache:il1 dl1
-cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
```


Notes...

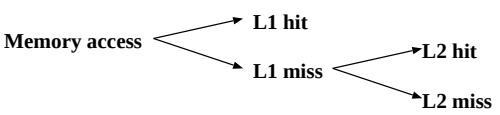
- **Q:** how would miss-rate vary with block size?
- **Q:** how might miss penalty for large blocks be reduced?
- **Q:** what happens with a large block size, when a write miss occurs?
- **Q:** Should the tag be the virtual address or the physical address?
- **Q:** Should the second-level cache always contain everything held in the first-level cache (the "multi-level inclusion" property)
- **Q:** How could one restructure a program to get better performance with a small cache?
- **Q:** What kind of program would have a very high rate of conflict misses with a two-way set-associative cache?

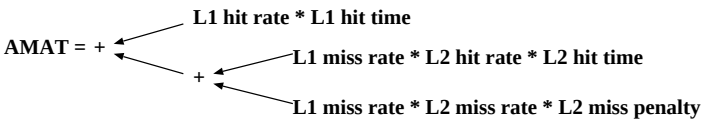
3.9 Average memory access time

The bottom line in memory system design is the average memory access time, AMAT. For a simple cache:

Average memory access time
= Hit time + Miss rate × Miss penalty

For a machine with a secondary cache:





AMAT is, of course, only part of the complete performance equation:

CPU time
= (CPU execution clock cycles + Memory stall clock cycles)
× Clock cycle time

- In a highly-dynamic CPU, instruction scheduling may reduce stalls due to memory accesses
- Further stalls may occur due to structural hazards, eg simultaneous instruction and data accesses, or peripheral activities

Caches: Summary

- SRAM is much faster than DRAM but too expensive and power-hungry for bulk storage
- Caches rely on spatial and temporal locality
- Design variables: capacity, associativity, block size, replacement policy and many more
- Write-through vs write-back
- Cache misses due to the three C's: Compulsory, Capacity, Conflict
- Average memory access time (AMAT) depends on applications's access pattern, hit time, miss rate, miss penalty