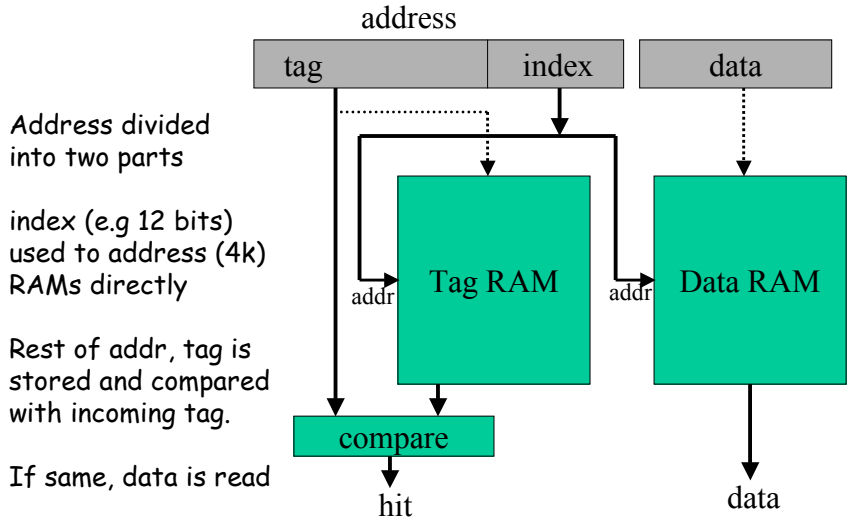
Cache Implementation

Ian Watson & Mikel Lujan
Advanced Processor Technologies Group

Cache Implementation

- Fully associative cache is logically what we want.
- But the storing of full (32 bit ?) addresses and the hardware comparison is expensive (and uses a lot of power)
- It is possible to achieve most of the functionality using ordinary small and fast (usually static) RAM memory.
- Few real caches are fully associative.

Direct Mapped Cache (1)



Direct Mapped Cache (2)

- Is really just a hash table implemented in hardware.
- Index / Tag could be selected from address in many ways.
- Most other aspects of operation are identical to fully associative.
- Except for replacement policy.

Direct Mapped Replacement

- New incoming tag/data can only be stored in one place - dictated by index
- Using LSBs as index exploits principle of spatial locality to minimize displacing recently used data.
- Much cheaper than associative (and faster?) but lower 'hit rate' (due to inflexible replacement strategy).

Cache Hit Rate

- Fraction of cache accesses which 'hit' in cache.
- Need > 98% to hide 50:1 ratio of memory speed to instruction speed
- Hit rates for instructions usually better than for data (more regular access patterns - more locality)

Set Associative Caches (1)

- A compromise!
- A set associative cache is simply a small number (e.g. 4) of direct mapped caches operating in parallel
- Now replacement strategy can be a bit more flexible
- We can choose any one of 4 - using LRU, cyclic etc.

Set Associative Caches (2)

- Hit rate gets better with increasing number of ways.
- Obviously higher no. of ways gets more expensive
- In fact N location fully associative cache is the same as single location N way set associative cache!

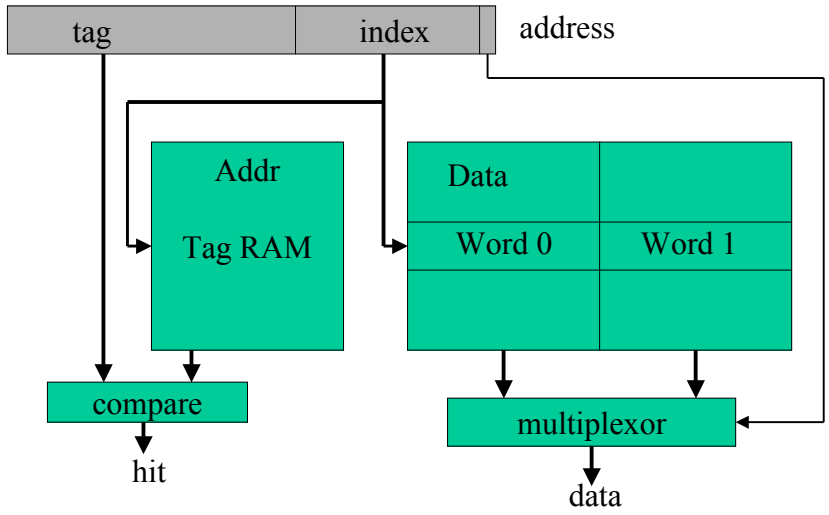
Cache Control Bits

- We have ignored detail of cache initialization.
- At some time it must start empty. We need a valid bit for each entry to indicate meaningful data.
- We also need a 'dirty' bit if we are using 'Write Back' rather than 'Write Through'

Exploiting spatial locality

- Storing and comparing the address or tag (part of address) is expensive.
- So far we have assumed that each address relates to a single data word.
- We can use a multiple word cache 'line' and store more data per address/tag
- Spatial locality suggests we will make use of it (e.g. series of instructions)

Direct Mapped cache - 2 word line



Multiple Word Line Size

- Now bottom bits of address are used to select which word
- Can be used with fully or set associative as well.
- Typical line size 16 or 32 bytes (e.g. 4 or 8 32 bit words)
- Transfer from RAM in 'blocks' which are equal to or less than line size - so line may not all be valid (need a valid bit for data too)

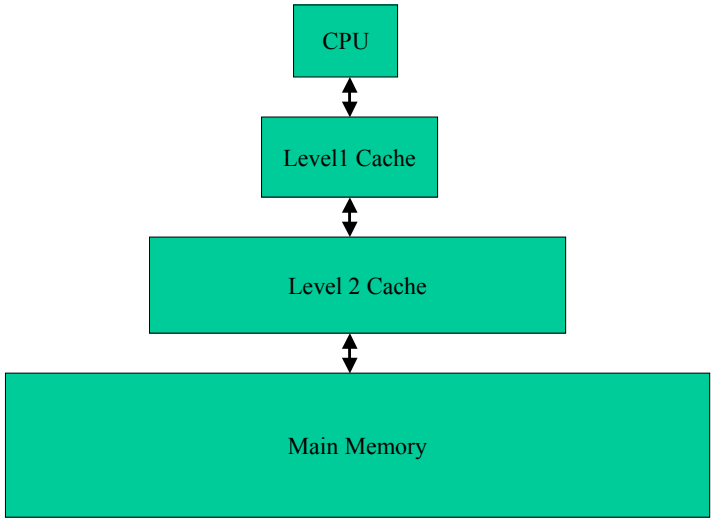
Separate Instruction & Data (I&D) Caches

- Instruction fetch every instruction
- Data fetch every 3 instructions
- Better bandwidth if we use separate caches
- Usually working in separate address areas
- Access patterns different - Inst. Access in serial sections - can use lower associativity
- Called 'Harvard' architecture

Multiple Level Caches (1)

- Bigger caches have lower miss rates
- As chips get bigger we can build bigger caches to perform better
- But bigger caches always run slower
- L1 cache needs to run at processor speed
- Instead put another cache (Level 2) between L1 and RAM

Multiple Level Caches



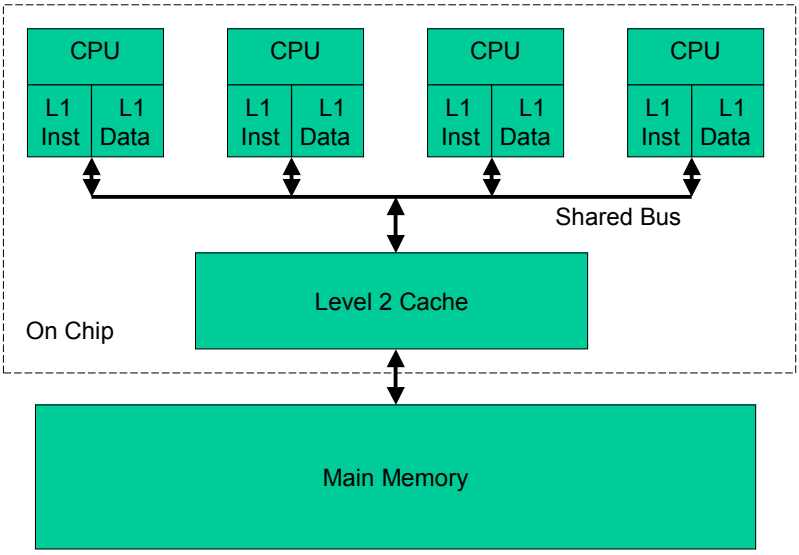
Multiple Level Caches (2)

- L2 cache is typically 16x bigger than L1
- L2 cache is typically 4x slower than L1 (but still 10x faster than RAM)
- If only 1 in 50 accesses miss in L1 and similar in L2 - only have to cover very small number of RAM accesses
- Not quite that easy but works well.

Multiple Level Caches (3)

- L2 is usually not split i.e. shared by L1 I&D
- Vital to performance of modern processors
- Intel Itanium & AMD ? even has L3 cache
- Replacement strategy and write policy obviously gets more complex

Typical Multi-Core Structure



Typical Multi-Core Structure

- Things to note
  - Cores have separate I & D caches
  - Shared Level 2 cache (I & D both stored there)
  - CPUs / Caches connected by a shared bus (i.e. common wires)
  - Can communicate with each other or L2 (more later)
  - CPU, Level1 caches and Level2 cache are all on the same 'chip'
  - Same chip means fast communication
  - Main memory is 'off chip' – slow communication

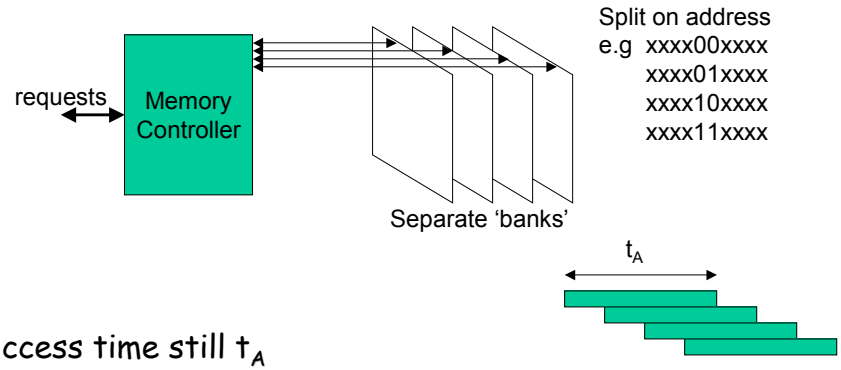
Memory (cache & main) Characteristics

- Memory has two important parameters
- ACCESS TIME (t_A)
  - If an address is presented to the memory t_A is the time delay before the data is available (or time it takes to write data)
- DATA RATE or BANDWIDTH
  - The rate (bits or bytes per second) at which data can be read or written to the memory
  - Might seem that this is just W/t_A bits/sec, where W is the width of the memory word – but if bandwidth matters, we can do better.

Interleaved Memory

- Both L2 and main memory are shown as single units
- They need to be shared between all cores
- But they can get too slow, particularly as we increase the number of cores.
- We have to accept the access time but very often it is the bandwidth which is the problem
- Although cores do share variables, very often they are accessing separate areas of data – we can exploit this.

Interleaved Memory



- Access time still t_A
- But up to 4 accesses within t_A (no bank conflict)
- Bandwidth is now $4W/t_A$ (max)
- Can use for (L2) cache or main memory