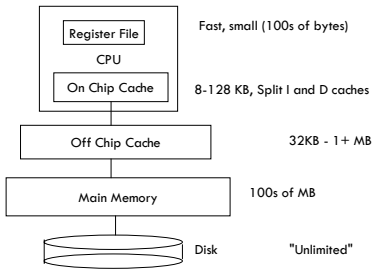
Memory Hierarchy

Introduction

- Problem: Suppose your processor issues one instruction per cycle and that 30% of the instructions are loads/stores.
  - How many memory references per instruction are there?
  - How much memory bandwidth do you require?
- Now what if the processor wishes to issue 4 instructions per cycle?
- Solution: Build a memory hierarchy.
  - One or more levels of cache in the CPU (fast, SRAM, $)
  - Maybe another level of cache outside of the CPU (slower, cheaper)
  - Main memory (slowest, cheapest)
- Trend: CPU speeds are increasing faster than memory access times are increasing...

Mem Hierarchy in Pictures



Caches

- Register file is not big enough to keep everything.
- Main memory is too far away -- it takes many cycles to access it.
- Put fast memory between the main memory and the registers: a cache
- When fetching an instruction (or performing a load) first check the cache.
  - If we find the address there, use that value.
  - If we don't find the address, fetch from memory and update the cache.
- When performing a store, first write it in the cache before updating main memory.
- Every current micro has at least 2 levels of cache.

Memory Technologies

- Technologies:

Type	Size	Access time	Relative speed (compared to reg access)	Cost
L1 Cache	16-64KB	nanoseconds	1-2	??
L2 Cache	64-256KB	10s of ns	5-10	$10/MB
Primary	256+MB	10s to 100s ns	10-100	$.25/MB
Secondary	10s of GB	10s of ms	1,000,000	$.01/MB

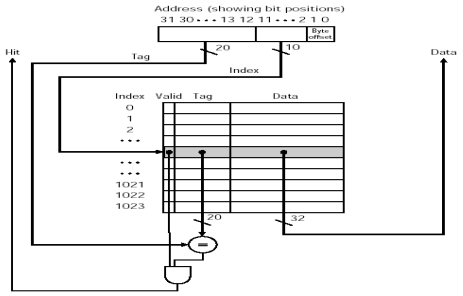
Locality

- Memory hierarchies work because programs exhibit locality:
  - *Temporal*: data (code) used in the past is likely to be used again in the future (eg. loops, stacks, etc.)
  - *Spatial*: data (code) close to the code that you are presently using is likely to be used in the near future (eg. traversing an array, sequences of instructions)

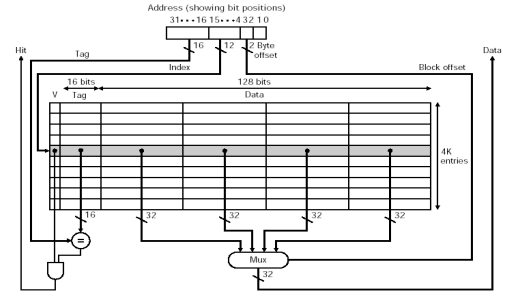
Terminology & Characteristics

- A *block* (or *line*) is the fundamental unit of data we transfer between two levels of the hierarchy.
- Where can a block be placed?
  - Depends on the organization.
- How do we find a block?
  - Each entry carries its own name (or *tag*).
- Which block do we replace when we bring in a new block?
  - One block (might) need to be kicked out. Depends on the *replacement policy*.
- What happens on a write?
  - Depends on the write policy.

A Direct-Mapped Cache



Spatial Locality: Bigger Blocks



Cache Performance

- Basic metric is *hit rate* (h)

$$\text{hit rate} = \frac{\text{number of memory references that hit in cache}}{\text{total number of memory references}}$$

- Miss rate = 1-h
- Now we can count *stall cycles*:

```
memory stall cycles = m * (total memory accesses) * miss penalty
```

- Now add this factor to basic equation:

```
CPU time = (CPU clock cycles + memory stall cycles) * cycletime
```


Performance 2

- Memory accesses per instruction depends on the mix, but is always > 1. Why?
- Example: gcc has 33% loads/stores, so it has 1.33 accesses/instruction
- Let's say our miss rate is 10% and the miss penalty is 20 cycles and our CPI = 1.8 (not counting memory stall cycles).
- What's the real CPI?
- Danger: building a processor with better CPI, but neglecting the memory performance...

Taxonomy of Misses

- The 3 Cs:
  - *Compulsory (cold) misses*: there will be misses the first time you touch a block of memory.
  - *Capacity misses*: the cache is not big enough to hold all the blocks you want.
  - *Conflict misses*: two blocks are mapped to the same location and there is not enough room in that location...

Design Parameters

- We can vary many parameters.
- The goal is to get the hit rate as high as possible, without making Tcache (cache access time) too big:
  - *Size*: Bigger caches = higher hit rates, but higher Tcache. Reduce capacity misses.
  - *Block size*: Larger blocks = higher hit rates, exploit spatial locality, but increase Tmem
  - *Associativity*: Smaller associativity = lower Tcache but lower hit rates
  - *Write policy*: later.
  - *Replacement policy*: later

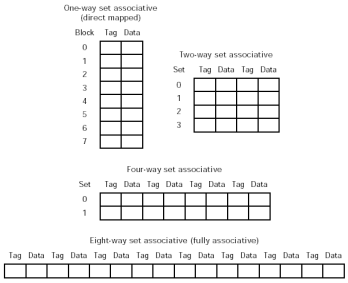
Block Size

- Block size is the number of bytes of data stored into one cache line.
- On a miss, a whole block is brought into the cache.
- Larger blocks have these advantages:
  - Decrease miss rate IF the program exhibits good spatial locality.
  - Increase transfer efficiency between cache and main memory.
  - Need fewer tags.
- ... and drawbacks:
  - Increase the latency of memory transfer.
  - Might bring unused data IF the program has poor spatial locality.
  - Increase conflict misses.

Associativity

- The mapping of memory locations to cache locations can be fully general to very restrictive.
- *Fully associative*:
  - A memory location can be mapped anywhere in the cache.
  - Doing a lookup is expensive.
- *Set associative*:
  - A memory location can map to a set (usually 2, 4, 8) locations.
- *Direct mapped*:
  - A memory location maps to just one location.
- Set associativity reduces conflict misses.

In Pictures



Replacement Policy

- On a read miss, we bring data into the cache.
- What if there's no room? Need to kick an item out.
- Direct mapped: easy, just replace the item...
- For set associative, we can replace any item in the set. Which is best:
  - Random
  - FIFO (the oldest one)
  - LRU (the least recently used one)
- For caches, replacement policy has minimal impact on performance...

Write Policy

- When we write a location, it's either in the cache (*write hit*) or not (*write miss*)
- On a write hit, we'll update the cached value (obviously) but what about main memory?
- Two choices:
  - Write through -- update memory right away.
  - Write back -- update memory only when replacing the block...

Write through

- The good:
  - Memory is always current (coherent); easier for I/O (see later)
  - Read misses don't result in writes to a lower level.
  - Easy to implement.
- The bad:
  - Writes occur at the speed of main memory (not the cache!)
  - Requires more memory bandwidth since every write goes to memory.

Write back

- Only update the entry in the cache. Only write main memory when we replace a block.
- This requires a *dirty bit* per block to indicate if the block has been written.
- On replacement, if the dirty bit is set, we need to write back the block, otherwise we don't care. Detail: reset the dirty bit when we bring in the new block!
- Good:
  - All writes occur at the speed of cache memory.
  - Less memory bandwidth needed.
- Disadvantages:
  - Coherency problems
  - Replacement can be expensive (incurring additional writes!)

Write Misses

- Again, there are choices:
  - Write around -- write only in memory (aka no-fetch)
  - Write allocate -- bring data into the cache and then write it.
- Note these policies are independent of the write hit policy!
- On write-allocate write-back, we need to write back the replaced block if dirty.
- On write-around write-back, we still need to write back dirty blocks on replacement.

Optimizations

- Sub-block placement:
  - For lines with multiple words, have one dirty and one valid bit per word.
  - Can read only individual words.
  - Only write back the dirty words.
- Write buffers:
  - A small (1-10 items) buffer between cache and main memory.
  - Each entry is a pair: <address> <data>
  - The stores to memory go through this buffer. The buffer is emptied while the CPU is working.
  - Still has to stall if the buffer fills up: when writes occur in bursts.
  - Why can the buffer be so small?

Split vs. Unified Caches

- Early caches were unified.
- RISCs require an instruction per cycles AND (possibly) a load/store.
- For this reason, modern machines split the cache into an I-cache and D-cache. This gives the illusion of two memories.
- Larger, off-chip caches are unified.
- I-caches can be simpler, because they are read-only

Coherency

- Most Disk I/O transfers data directly from disk to memory.
- We'll use this terminology:
  - read: transfer from disk to memory
  - write: transfer from memory to disk
- Reads:
  - Write through and write back: data transferred during the read must invalidate corresponding cache entries.
- Writes:
  - Write-back: the cached data may not be coherent with memory before a write.
  - Need to flush the cache.