Handling hazards: the hardcore edition

Multi-cycle execution
  - Microarchitectural support
  - Code scheduling

Branch prediction
  - Static vs dynamic
  - 2-bit predictor
  - 2-level correlating predictor
Announcements

- Coursework 1 is out
  - Due next Monday, Feb 16 @ 4pm

- Tutorials happenin’ this week
  - Study advice: look at the questions ahead of time!

- Next week:
  - Innovative learning week. No lectures!
Ideally one would desire an indefinitely large memory capacity such that any particular … word would be immediately available … we are … forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

A. W. Burks, H. H. Goldstine, and J. von Neumann - 1946
The Memory Gap

Bottom-line: memory subsystem design increasingly important
Principle of Locality

- **Temporal Locality:**
  - A recently accessed memory location (instruction or data) is likely to be accessed again in the near future

- **Spatial Locality:**
  - Memory locations close to a recently accessed location are likely to be accessed in the near future

- **Why?**
  - Instruction reuse: loops, functions
  - Data **working sets**: arrays, temporary variables, structures

- **Bottom-line:** small, fast caches backed up by larger, slower memories give the impression of a single, large, fast memory
Memory Hierarchy

upper levels

- faster
- smaller
- more expensive

512B <1ns

8-64KB 1-2ns

1-40MBs 10-20ns

lower levels

- slower
- bigger
- cheaper

1-100GBs 100ns

Flash: <1TB, ~10 μs

>1TB 2-5ms

1ns (1GHz)

processor

registers

1st level Cache (L1)

2nd level Cache (L2)

Main memory

Disk

Explicitly managed by the user program (compiler)

Transparently managed by the cache and memory controllers

Transparently managed by OS virtual memory manager
Memory Hierarchy Issues

- **Block size:** smallest unit that is managed at each level
  - E.g., 64B for cache lines, 4KB for memory pages

- **Block placement:** Where can a block be placed?
  - E.g., direct mapped, set associative, fully associative

- **Block identification:** How can a block be found?
  - E.g., hardware tag matching, OS page table

- **Block replacement:** Which block should be replaced?
  - E.g., LRU, random

- **Write strategy:** What happens on a write?
  - E.g., write-through, write-back, write-allocate

- **Inclusivity:** whether next lower level contains all the data found in the current level
  - Inclusive, non-inclusive
Cache Block Placement

Memory

Block: 0 1 12 31

Cache

Fully associative:
block 12 can go anywhere in the cache

Cache

Direct mapped:
block 12 can only go into block 4
(12 mod 8)

Cache

Set associative:
block 12 can go anywhere in set 0
(12 mod 4)
Cache Associativity

- **Fully associative**
  - The block from the lower level can go into any block frame in the cache
  - Most flexible approach → lowest miss rate
  - Must search the whole cache to find the block → increased access time and high power consumption

- **Direct mapped**
  - The block from the lower level can only go into one frame in the cache
  - Simplest approach to implement
  - Cache can fill up unevenly → increased miss rates

- **Set associative**
  - Split the cache into groups of m blocks (sets) → m-way set associative
  - The block from the lower level can only go into one set, but then within that set it can go anywhere
  - Good compromise: 2 or 4-way set associative
Cache Block Identification

- Every block is identified by a name or tag, which is part of the memory address.
- Block tag is stored alongside the block data in the cache.
- Block tags in the cache are compared with the tag of the requested block → often in parallel, for speed.
- Block tag from memory address:

Full memory address: | Tag | Index | Block offset
---|---|---|---
- Data address: the address of the byte being referenced → 32 bits for MIPS
- Offset: the byte within the block; e.g., 6 bits for a 64B block
- Index: the set where the block can be found; e.g., 8 bits for a 4-way 64KB cache
- Tag: the “ID” of the block; e.g., 32-8-6=18 bits
Address Mapping Example

- Cache: 32 KBytes, 2-way, 64 byte lines
- Virtual address: 32 bits

Example: \((150000)_{10} = (0000 \ 0000 \ 0000 \ 0010 \ 0100 \ 1001 \ 1111 \ 0000)_{2}\)

  - Byte offset
    64 bytes → 6 bits ⇒ 11 0000

  - Index: \(32K/64 = 512\) lines in the cache
    \(512/2 = 256\) sets in the cache
    \(150000/64=2343\) (i.e., line 2343)
    \(2343 \mod 256 = 39\) (i.e., set 39)
    \(256\) sets → 8 bits ⇒ 00 1001 11 = 39

  - Tag:
    \(2343/256=9\)
    \(0000 \ 0000 \ 0000 \ 0010 \ 01\)
Cache Organization: Direct mapped

H&P
(2e)
Fig. 5.5 (Alpha 21064)
Cache Organization: 2-way set associative

Figure 5.7 (Alpha 21264)
Cache Block Replacement

- To bring a new block in the cache an old one must be evicted.
- Direct mapped caches: there is only one choice of block to evict.
- Associative caches:
  - Random: select any block in the set randomly.
  - Least-recently-used (LRU): select the block that has not been used for the longest period of time → works well in practice because of the principle of locality.
  - Ideal: select the block that will not be used for the longest period of time → cannot be implemented in practice.
Cache Write Strategies

What happens to the next lower level in the memory hierarchy?

- **Write through**: write to lower level as cache is modified
  - Generates more traffic
  - Writes will perform at the speed of the lower level of memory hierarchy
  - Lower level is kept **consistent** with cache (particularly important for multi-processors)

- **Write back**: only write to lower level when the block is evicted
  - Generates less traffic
  - Writes will perform at the speed of the cache
  - Lower level can have stale data for some time (**cache-coherency** problem)
Cache Write Strategies

What happens if the block is not found in the cache?

- **Write allocate**: bring the block into the cache and write to it
  - Good if block will soon be used by another memory access (locality)
  - Usually used with write back

- **Write no-allocate**: do not bring block into cache and modify data in the lower level
  - Good if no memory access to the same block occur in the near future
  - Usually used with write through