CMSC 411
Computer Systems Architecture
Lecture 8
Basic Pipelining, cont., & Memory Hierarchy

Alan Sussman
als@cs.umd.edu

Administrivia

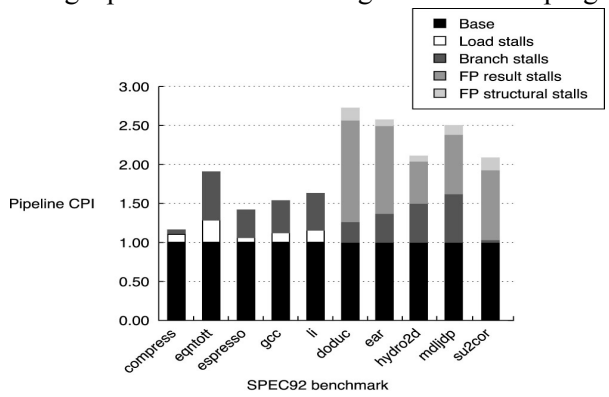
- **Homework #1 returned today**
  - **solutions posted on password protected web page**
- **Homework #2 posted and due next Tuesday**
- **Read Appendix B of H&P**
- **First project, on basic pipelining, posted by tomorrow**

R4000 pipeline performance

- **4 major causes of pipeline stalls**
  - load stalls **– from using load result 1 or 2 cycles after load**
  - branch stalls **– 2 cycles on every taken branch, or empty branch delay slot**
  - FP result stalls **– RAW hazards for an FP operand**
  - FP structural stalls **– from conflicts for functional units in FP pipeline**

SPEC92 benchmarks

Assuming a perfect cache – 5 integer and five FP programs



Dynamically scheduled pipelines

- **We'll cover this, and the scoreboard technique, in Unit 5**

Pitfalls

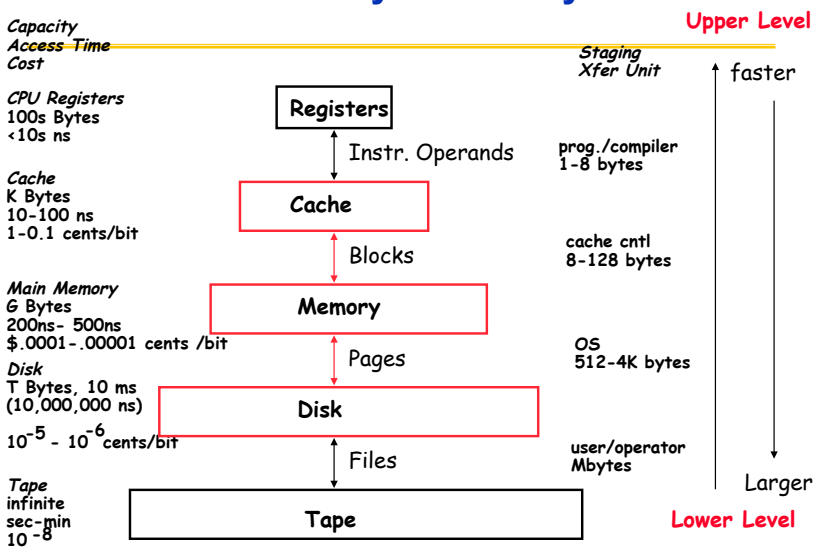
- Unexpected hazards do occur …
  - **for example, when a branch is taken before a previous instruction finishes**
- Extensive pipelining can slow a machine down, or lead to worse cost-performance
  - **more complex hardware can cause a longer clock cycle, killing the benefits of more pipelining**

Pitfalls (cont.)

- A poor compiler can make a good machine look bad
  - **compiler writers need to understand the architecture in order to**
    - » **optimize efficiently and**
    - » **avoid hazards**
  - **better to eliminate useless instructions, than make them run faster**

MEMORY HIERARCHY

Levels of the Memory Hierarchy

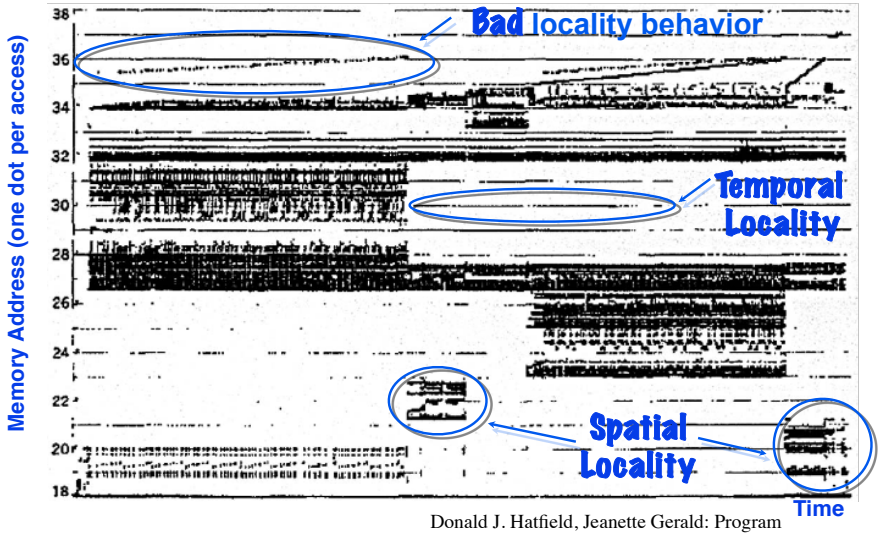


The Principle of Locality

- The Principle of Locality:
  - Program accesses a relatively small portion of the address space at any short period of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 15-20 years, HW has relied on locality to improve overall performance

It is a property of programs that is exploited in machine design.

Programs with locality cache well ...



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Issues to consider

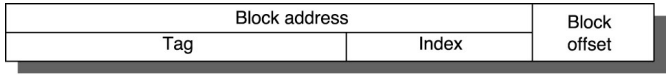
- How big should the fastest memory (cache memory) be?
- How do we decide what to put in cache memory?
- If the cache is full, how do we decide what to remove?
- How do we find something in cache?
- How do we handle writes?

First, there is main memory

- **Jargon:**
  - **frame address – which page?**
  - **block number – which cache block?**
  - **contents – the data**

Then add a cache

- **Jargon: Each address of a memory location is partitioned into**
  - **block address**
    - » **tag**
    - » **index**
  - **block offset**

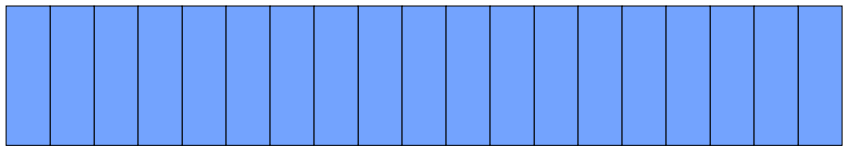


How does cache memory work?

- **The following slides discuss:**
  - **what cache memory is**
  - **three organizations for cache memory**
    - » **direct mapped.**
    - » **set associative**
    - » **fully associative**
  - **how the bookkeeping is done**
- **Important note: All addresses shown are in octal. Addresses in the book are usually decimal.**

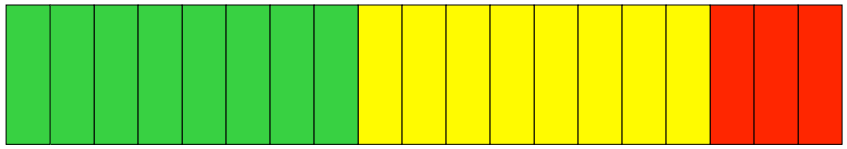
What is cache memory? Main memory first

Main memory is divided into (cache) blocks.
Each block contains many words (32-256 common now).



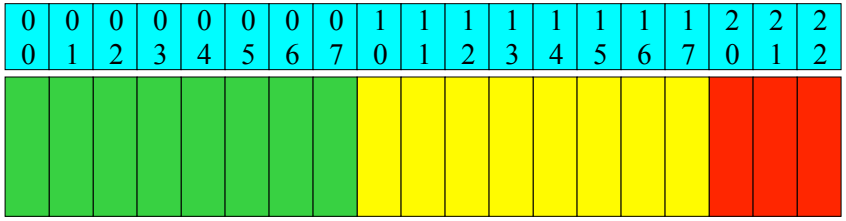
Main memory

Blocks are grouped into frames (pages), 3 frames in this picture.



Main memory (cont.)

Blocks are addressed by their frame number, and their block number within the frame.



Cache memory

Cache has many, MANY fewer blocks than main memory, each with

a **block number**, 0 1 2 3 4 5 6 7

a **memory address**, 10 21 42 53 74 25 16 77

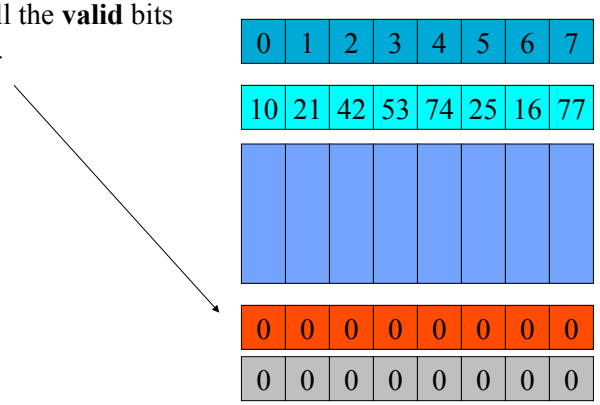
data,

a **valid** bit, 0 0 0 0 0 0 0 0

a **dirty** bit. 0 0 0 0 0 0 0 0

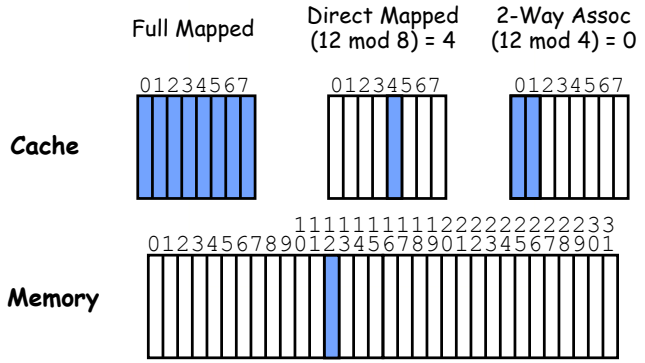
Cache memory (cont.)

Initially, all the **valid** bits set to zero.



Where can a block be placed?

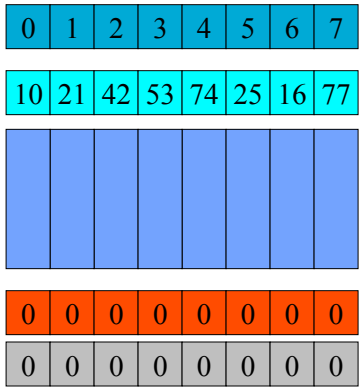
- **Block 12 placed in 8 block cache:**
  - **Fully associative, direct mapped, 2-way set associative**



Cache memory (cont.)

Suppose want to load block 14 (octal) from memory into cache.

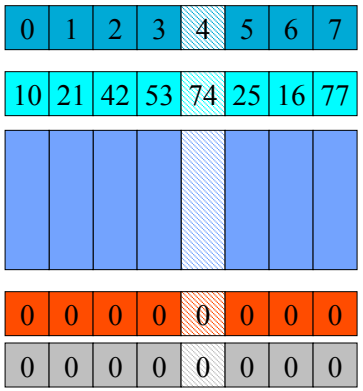
Three ways to organize cache
- direct mapped
- set associative
- fully associative



Direct mapped cache

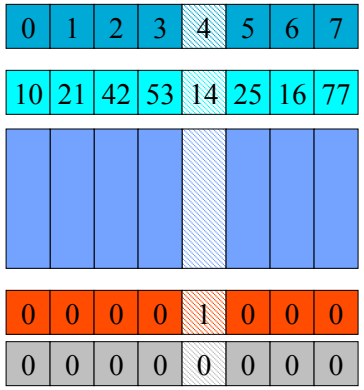
In **direct mapped cache,** block 14 can only be put in the cache block with address 4.

So the cache will no longer hold the block with memory address 74.



Direct mapped cache (cont.)

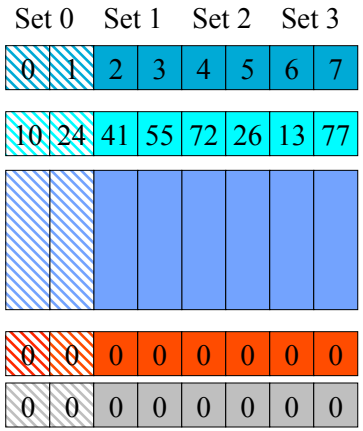
After the load, the contents look like this.



Set associative cache

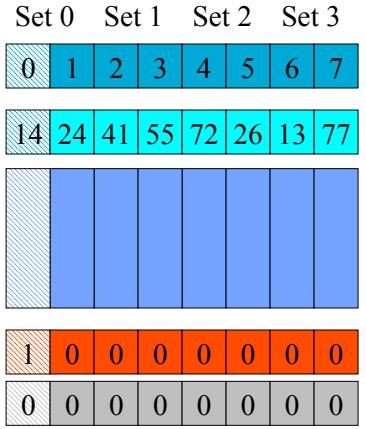
In **set associative cache**, each memory block can be put in any of a set of possible blocks in cache.

For example, if divide cache into 4 sets, block 14 can be put in any block in Set 0 (since last two bits of 14 octal are zero).



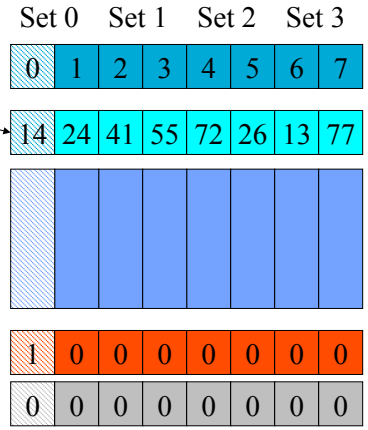
Set associative cache (cont.)

So after loading the block, cache memory might look like this.



Set associative cache (cont.)

Note that the last two bits of the memory block's address always match the set number, so do not need to be stored. This part of the address is called the **index**. The higher order bits are stored, and are called the **tag.**
In these pictures, both index and tag shown.



Recall:

Set associative cache replacement

- **Which entry in the set to replace?**
- **Three common choices:**
  - **Replace an eligible *random* block**
  - **Replace the least recently used (LRU) block**
    - » **can be hard to keep track of, so often only approximated**
  - **Replace the oldest eligible block (First In, First Out, or FIFO)**

Data cache replacement example

SPEC2000, in misses per 1000 instructions

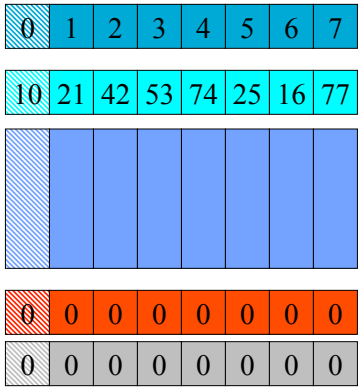
Set associativity

	Two-way			Four-way			Eight-Way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Fully associative cache

In **fully associative** cache, memory blocks may be stored anywhere.

So block 14 might be put in the first available block -- one with **valid** = 0.



Fully associative cache (cont.)

With this result.