Notes and Updates

- Office hours for Dr. Simon: Tues 3:30-5 and Wed 12-1:30, Thurs before final 9-12 and 2-5:30
- Quiz Thursday! (no more hw)
- Today:
  - Stores and caches
  - Alignment
  - Multilevel caches
  - Types of Cache Misses
  - Virtual Memory
- Thurs: Virtual Memory
- Review session: Tuesday Center 109 6:30-7:50pm
- Consent Forms, Surveys on Web System
- Questions?

A set associative cache

128 KB cache, 4-way set-associative, 8-byte blocks

This picture doesn't show the "most recent" bits (?? bits per set)
Cache Performance (do examples on pg 494-495)

- Instruction cache miss rate of 4%, data cache miss rate of 9%, BCPI = 1.0, 20% of instructions are loads and stores, miss penalty = 10 cycles, TCPI = ?

\[
MCPI = \%\text{InstMemRefs} \times \text{InstMissRate} \times \text{InstMissPenalty} + \%\text{DataMemRefs} \times \text{DataMissRate} \times \text{DataMissPenalty}^3
\]

Cache Performance: For your practice

- Unified cache, 25% of instructions are loads and stores, BCPI = 1.2, miss penalty of 10 cycles. If we improve the data miss rate from 10% to 4% (e.g. with a larger cache), how much do we improve performance?

- BCPI = 1, miss rate of 20% for data, 2% for instructions, 20% loads, miss penalty 20 cycles (both instruction and data cache). What is the speedup from doubling the cpu clock rate?
• Can you calculate a “real” Base CPI?
  - See some of the last week’s recommended problems
  - Things like: Suppose you have 1000 instructions composed of a repeated pattern of a load followed by and add that depends on that load. Additionally, the next load depends on the add directly before it.
  - What is the Base CPI? (ie, CPI for our 5 stage pipeline with the forwarding that we developed).

Can you do?

• Can you calculate a “real” Base CPI?
  - Given a code with a loop - and a misprediction rate for that loop. (ie the branch misses 50% of the time).
Dealing with Stores

- Stores must be handled differently than loads, because...
  - they don’t necessarily require the CPU to stall
    - Who “needs” the value?

- they change the content of cache
  Creates a memory consistency question ... how do you ensure memory gets the correct value?

Policy decisions for stores

- Do you keep memory and cache identical?
  - write-through cache: all writes go to both cache and main memory
  - write-back cache: writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
Policy decisions for stores

- Do you make room in cache for store miss?
  - write-allocate: on a store miss, bring target line into the cache.
  - write-around: on a store miss, ignore cache

Dealing with stores

- On a store hit, write the new data to cache.
  - In a cache, write the data immediately to memory.
  - In a cache
    - Mark the line as dirty.
    - On any subsequent cache miss in a write-back cache, if the line to be replaced in the cache is dirty, write it back to memory.

- On a store miss,
  - In a cache,
    - Initiate a cache block load from memory.
  - In a cache,
    - Write directly to memory.
Cache Alignment

- A cache line is all the data whose addresses share the tag and index.
  - Example: Suppose offset is 5 bits,
    - Bytes 0-31 form the first cacheline
    - Bytes 32-63 form the second, etc.
    - When you load location 40, cache gets Bytes 32-63
- This results in
  - no overlap of cache lines
  - easy to find if address is in cache (no additions)
  - easy to find the data within the cache line
- Think of memory as organized into cacheline sized pieces (because in reality, it is!)

Can a word overlap two cache lines?
 i.e. must all integer addresses end in 00?

- Depends on the architecture ...
  - Some require words to be word-aligned ...
    - and double words to be double-word aligned
    - Every load and store is to a single cacheline
  - Others allow data to span cache lines
    - Can be two cache misses for a single reference!
    - Requires more shifting logic
    - But allows more compact data structures.
Two memory allocation schemes

- No alignment required

```
struct{
    char valid;
    int tag;
} cache_line;

cache = new cache_line[1024];
```

- Alignment required

```
tag 1 word data
M[0-3]  
M[4-7]  
M[8-11] 
M[12-15]
M[16-19] 
M[20-23] 
M[24-27] 
M[28-31]
```

Multilevel Caches: A Performance Necessity

- Page 505 in the book, a good story,
- Only a single L1 cache, accesses at speed of processor, only misses 2%** of time
  - Increases CPI from 1.0 to 11.0!!!
  - Equivalent of making a processor 11 times slower! (instead of 5GHz → ~500MHz!)
- So - we need multiple levels of cache
  - L1 in your cycle time limit
  - L2 in a reasonable number of cycles (10-100)
What does a CPI of 11 look like?

Calculating CPI with some cache miss rates: You need to be able to do this

- Processor: 5GHz, .2ns
- L1 accesses in 1 cycle, time to access main memory 100ns (500 cycles)
- Miss rate 2%
  - That's counting all instructions!
  - What if I say that 0.5% of all instruction memory accesses miss and 5% of all data memory accesses miss (and 30% of all instructions are lds/sts)
A Level 2 cache

- Add a secondary cache that takes 5ns to access (25 cycles)
  - 2% still miss in L1
  - Only 0.5% of all accesses miss in L2 (and go to MM)
- CPI = 1 +

Talking about Miss Rates (not terribly standard)

- Global
  - The denominator is the number of accesses made by the pipeline OVERALL
  - The numerator is the number of accesses that miss all the way to MAIN MEMORY
  - Global miss rate for this code is...

- Local
  - The denominator is based on the number of accesses that are made to THIS LEVEL
  - The numerator is the number of accesses that miss in THIS LEVEL
  - L1 local miss rate, L2 local miss rate, L3 local miss rate
### Three types of cache misses (page 543)

- **Compulsory misses**
  - number of misses needed to bring every cache line referenced by program into an infinitely large cache.

- **Capacity misses**
  - number of misses in a fully associative cache of the same size as the cache in question minus the compulsory misses.

- **Conflict misses**
  - number of misses in actual cache minus number there would be in a fully-associative cache of the same size.

<table>
<thead>
<tr>
<th>valid</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Three types of cache misses

Total misses = (Compulsory + Capacity + Conflict) misses

- **Example: 16-byte, direct-mapped, 1 word per cacheline**
  - Reference sequence: 4, 8, 12, 4, 8, 20, 4, 8, 20, 24, 12, 8, 4

<table>
<thead>
<tr>
<th>Compulsory misses:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Capacity misses:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conflict misses:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- **Example 16-byte, direct-mapped, 2 words per cache line**
  - What is main difference?

<table>
<thead>
<tr>
<th>Compulsory misses:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Capacity misses:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conflict misses:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
So, then, how do we decrease...

- Compulsory misses?
- Capacity misses?
- Conflict misses?

It’s not all about Big O: page 508
- Quicksort versus Radix Sort (Radix has better Big O)
Virtual Memory:
Or where do I find that PC anyways?

- When you compile, jumps are filled in with actual addresses to jump to
- But how does the compiler know where your code will be assigned in memory when it is run?

Other reasons to have virtual memory

- Managing lots of different programs running at the same time
  - When put together, perhaps they exceed the size of memory
  - Protecting one program’s space from another’s
- Supporting REALLY large programs that won’t fit in main memory all at once
  - Historical or for embedded, limited memory processors
- Breaking the “link” between an ISA decision (all memory addresses will be 32 bits) and the real size of main memory
Assembly uses Virtual Addressing

- Each PC address and memory address used by a program is a virtual address
  - That way the compiler can use any range of addresses it wants
- At runtime, the OS assigns a program into a particular physical memory space
  - Which can provide protection, if you try to access out off your space...
- But now EVERY ACCESS to memory must be translated

The basic idea (page 7.19)
The basic idea (page 7.19)

- Physical memory is broken into a number of pages
  - A range of P addresses that map to a similar range of V addresses
  - Reduces bookkeeping
- A page may be in PM or NOT (on disk)
- LIKE A ____________

Physical addresses
In REAL main memory

Disk

Virtual Address Translation (pg 513)

- **Cache**
  - **tag**
  - **index**
  - **offset**
- **VA**
- **PA**

32 bits

However many Bits you need to Access your memory size
Block/Line size vs page size

- Page size large enough to “cover” or “amortize” time to swap a page
  - 4-16KB typical
- Page fault (miss) EXPENSIVE – millions of cycles
  - Use software controlled fully associative
  - Spend some TIME in software alg to get the best possible replacement scheme
- Write back!
  - Pages on disk will be out of date
  - Ever unplugged your computer while it was running?
  - What happens on a “save” from word?

Hidden detail:
address translation mechanism

- A Page Table: indexed by VP number
Hidden detail: The translation mechanism

- Since any virtual page # can map to any physical page # (ie fully assoc)
  - How do we get at it FAST?
  - We need to do this in significantly less than one cycle! (IF (PC) and MEM (ld/st) stages)