Review Problem 35

* What should we do to this code to run it on a CPU with delay slots?*

```assembly
AND   X0, X1, X2
ORRI  X0, X0, #7
ADD   X3, X4, X5
LDUR  X6, [X3, #0]
CBNZ  X6, FOO
B     BAR
```
The Solution

By taking advantage of the principle of locality:

Provide as much memory as is available in the cheapest technology.
Provide access at the speed offered by the fastest technology.

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Cache</th>
<th>Main Memory</th>
<th>Disk Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>1 cycle</td>
<td>1-7 cycles</td>
<td>100 cycles</td>
<td>10,000 cycles</td>
</tr>
<tr>
<td>Capacity</td>
<td>1x (norm.)</td>
<td>64-4Kx</td>
<td>4Mx</td>
<td>1Gx</td>
</tr>
</tbody>
</table>
Cache Terminology

**Block** – Minimum unit of information transfer between levels of the hierarchy
  
  Block addressing varies by technology at each level
  
  Blocks are moved one level at a time

**Upper vs. lower** level – “upper” is closer to CPU, “lower” is further away

**Hit** – Data appears in a block in that level
  
  **Hit rate** – percent of accesses hitting in that level
  
  **Hit time** – Time to access this level
    
    Hit time = Access time + Time to determine hit/miss

**Miss** – Data does not appear in that level and must be fetched from lower level
  
  **Miss rate** – percent of misses at that level = (1 – hit rate)
  
  **Miss penalty** – Overhead in getting data from a lower level
    
    Miss penalty = Lower level access time + Replacement time + Time to deliver to processor
    
    Miss penalty is usually MUCH larger than the hit time
Cache Access Time

Average access time
  Access time = (hit time) + (miss penalty)x(miss rate)

  Want high hit rate & low hit time, since miss penalty is large

Average Memory Access Time (AMAT)
  Apply average access time to entire hierarchy.
Cache Access Time Example

<table>
<thead>
<tr>
<th>Level</th>
<th>Hit Time</th>
<th>Hit Rate</th>
<th>Access Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>1 cycle</td>
<td>95%</td>
<td>$1 + 0.05 \times 65 = 65$</td>
</tr>
<tr>
<td>L2</td>
<td>10 cycles</td>
<td>90%</td>
<td>$10 + 0.1 \times 550 = 65$</td>
</tr>
<tr>
<td>Main Memory</td>
<td>50 cycles</td>
<td>99%</td>
<td>$50 + 0.01 \times 50,000 = 550$</td>
</tr>
<tr>
<td>Disk</td>
<td>50,000 cycles</td>
<td>100%</td>
<td>$50,000$</td>
</tr>
</tbody>
</table>

Note: Numbers are \textit{local} hit rates – the ratio of access that go to that cache that hit (remember, higher levels filter accesses to lower levels)
Handling A Cache Miss

Processor expects a cache hit (1 cycle), so no effect on hit.

Instruction Miss
1. Send the original PC to the memory
2. Instruct memory to perform a read and wait (no write enables)
3. Write the result to the appropriate cache line
4. Restart the instruction

Data Miss
1. Stall the pipeline (freeze following instructions)
2. Instruct memory to perform a read and wait
3. Return the result from memory and allow the pipeline to continue
Exploiting Locality

Spatial locality
Move blocks consisting of multiple contiguous words to upper level

Temporal locality
Keep more recently accessed items closer to the processor
When we must evict items to make room for new ones, attempt to keep
more recently accessed items
Cache Arrangement

How should the data in the cache be organized?

Caches are smaller than the full memory, so multiple addresses must map to the same cache “line”

**Direct Mapped** – Memory addresses map to particular location in that cache

**Fully Associative** – Data can be placed anywhere in the cache

**N-way Set Associative** – Data can be placed in a limited number of places in the cache depending upon the memory address
Direct Mapped Cache

4 byte direct mapped cache with 1 byte blocks
Optimize for spatial locality (close blocks likely to be accessed soon)

Memory Address

Cache Address

MA % Cache size = Cache Address