CSCI 654 – Advanced Computer Architecture
Homework 2

Due: September 26, 2016

Alexander Powell

1. - **Determine the number of sets in the cache:**
   Since the size of the cache is S and the block size is B, then the number of blocks in the cache is S/B. Since the cache is A-way set associative, then the number of sets in the cache can be defined as $\dfrac{S}{B \times A}$.
   - **Determine the number of index bits in the address:**
   Since we have already determined the number of sets in the cache to be $\dfrac{S}{B \times A}$, then the number of index bits is the base 2 logarithm of the number of sets, or:
   $$\log_2 \left(\frac{S}{B \times A} \right).$$
   - **Determine the number of bits needed to implement the cache:**
   Again, we can write the number of blocks in the cache as S/B. For each of these blocks we will have 1 valid bit, 1 byte (8 bits) of data per block, and some number of tag bits. The number of bits in the tag is the computer's address size minus the index bits (from above) and b. So, the number of tag bits is equal to $k - \log_2 (\frac{S}{B \times A}) - b$. Putting this altogether we get
   $$\frac{S}{B} \times \left(1 + k - \log_2 \left(\frac{S}{B \times A} \right) - b + 8 \times B \right).$$
   Since we know $b = \log_2 B$, then we can substitute to get the following:
   $$\frac{S}{B} \times \left(1 + \underbrace{k - \log_2 \left(\frac{S}{B \times A} \right) - \log_2 B} + 8 \times B \right).$$
   By just looking at the underlined section we can simplify with the following operations.
   $$k - \log_2 \left(\frac{S}{B \times A} \right) - \log_2 B$$
   $$= k - \left(\log_2 S - \log_2 \left(B \times A \right) \right) - \log_2 B$$
   $$= k - \log_2 S + \log_2 B + \log_2 A - \log_2 B$$
   $$= k - \left(\log_2 S - \log_2 A \right)$$
   $$= k - \log_2 \left(\frac{S}{A} \right)$$
   Now we can substitute this back into what we had originally to get:
   $$\frac{S}{B} \times \left(1 + k - \log_2 \left(\frac{S}{A} \right) + 8 \times B \right) \text{ bits needed to implement the cache.}$$

2. **3.1: Explain in a short paragraph why cache size and VM page size are interdependent.**

   Cache size and virtual memory page size are interdependent because cache line entries point to locations in memory. For example, a virtual page number might be mapped through the cache to get the physical address. This is called the TLB, or translation lookaside buffer, which is associative. When the cache size is larger than the virtual memory page size, cache aliasing can occur. Cache aliasing occurs when several different virtual addresses refer to the same physical address. To solve this problem, when performing a write, the lookup of the virtual cache and TLB should be done in parallel. Also, one should use a physical address from the TLB to check for aliases.

3. **What would be the baseline performance (in cycles per loop iteration) of the code sequence if no new instruction's execution could be initiated until the previous instruction's execution had completed?**

   First, there are 11 instructions to be executed (since we are told that the branch is taken), so there are at least 11 cycles per loop iteration. In addition, we have to account for the latencies associated with some instructions. There are two loads, so that's 8 more cycles, 1 store (1 cycle), 1 divide (12 cycles), 1 multiplication (5 cycles), 2 adds (1 cycle each), and 1 branch (1 cycle). Adding all of this up we get:

   $$11 + 8 + 1 + 12 + 5 + 2 + 1 = 40 \text{ cycles per loop iteration.}$$

4. **3.2: How many cycles would the loop body in the code sequence in Figure 3.48 require if the pipeline detected true data dependences and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? Show the code with ¡stall¿ inserted where necessary to accommodate stated latencies.**

```
Loop:   LD              F2, 0(Rx)
        <stall>  // 4 cycles for LD
        <stall>
        <stall>
        <stall>
        DIVD            F8, F2, F0
        MULTD           F2, F6, F2
        LD              F4, 0(Ry)
        <stall>  // 4 cycles for LD
        <stall>
        <stall>
        <stall>
        ADDD            F4, F0, F4
        <stall>  // Still counting 12 cycles for DIVD latency
        <stall>
        <stall>
        <stall>
        <stall>  // Finished 12th cycle for DIVD.  Can now access F8
        ADD             F10, F8, F2
        ADDI            Rx, Rx, #8
        ADDI            Ry, Ry, #8
        SD              F4, 0(Ry)
        SUB             R20, R4, Rx
        BNZ             R20, Loop
        <stall>  // Latency +1 for branch
```

   By only stalling on true data dependencies, we are able to reduce the number of cycles per loop iteration down to 25.