Sorry: This might take a while because of the applet loading!!!

Interaction Policies with Main Memory

Reads dominate processor cache accesses. All instruction accesses are reads, and most instructions do not write to memory. The block can be read at the same time that the tag is read and compared, so the block read begins as soon as the block address is available. If the read is a miss, there is no benefit - but also no harm; just ignore the value read.

The read policies are:
- ***Read Through -*** reading a word from main memory to CPU
- ***No Read Through -*** reading a block from main memory to cache and then from cache to CPU

Such is not the case for writes. Modifying a block cannot begin until the tag is checked to see if the address is a hit. Also the processor specifies the size of the write, usually between 1 and 8 bytes; only that portion of the block can be changed. In contrast, reads can access more bytes than necessary without a problem.

The **write policies** on write *hit* often distinguish cache designs:
- ***Write Through*** - the information is written to both the block in the cache and to the block in the lower-level memory.

Advantage:
- read miss never results in writes to main memory
- easy to implement
- main memory always has the most current copy of the data (consistent)

Disadvantage:
- write is slower
- every write needs a main memory access
- as a result uses more memory bandwidth

- ***Write back*** - the information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced. To reduce the frequency of writing back blocks on replacement, a ***dirty bit*** is commonly used. This status bit indicates whether the block is dirty (modified while in the cache) or clean (not modified). If it is clean the block is not written on a miss.

Advantage:
- writes occur at the speed of the cache memory
- multiple writes within a block require only one write to main memory
- as a result uses less memory bandwidth

Disadvantage:
- harder to implement

- main memory is not always consistent with cache
- reads that result in replacement may cause writes of dirty blocks to main memory

There are two common options on a write *miss*:

- ***Write Allocate*** - the block is loaded on a write miss, followed by the write-hit action.
- ***No Write Allocate*** - the block is modified in the main memory and not loaded into the cache.

Although either write-miss policy could be used with write through or write back, **write-back** caches generally use **write allocate** (hoping that subsequent writes to that block will be captured by the cache) and
write-through caches often use **no-write allocate** (since subsequent writes to that block will still have to go to memory).

Table 1 shows all possible combinations of interaction policies with main memory on write, the combinations used in practice are in bold case.

Write hit policy	Write miss policy
Write Through	Write Allocate
Write Through	**No Write Allocate**
Write Back	**Write Allocate**
Write Back	No Write Allocate

Table 1. Possible combinations of interaction policies with main memory on write.

Write Through with Write Allocate:

- on hits it writes to cache and main memory
- on misses it updates the block in main memory and brings the block to the cache
- Bringing the block to cache on a miss does not make a lot of sense in this combination because the next hit to this block will generate a write to main memory anyway (according to Write Through policy)

Write Through with No Write Allocate:

- on hits it writes to cache and main memory;
- on misses it updates the block in main memory not bringing that block to the cache;
- Subsequent writes to the block will update main memory because Write Through policy is employed. So, some time is saved not bringing the block in the cache on a miss because it appears useless anyway.

Write Back with Write Allocate:

- on hits it writes to cache setting "dirty" bit for the block, main memory is not updated;
- on misses it updates the block in main memory and brings the block to the cache;
- Subsequent writes to the same block, if the block originally caused a miss, will hit in the cache next time, setting dirty bit for the block. That will eliminate extra memory accesses and result in very efficient execution compared with Write Through with Write Allocate combination.

Write Back with No Write Allocate:

- on hits it writes to cache setting “dirty” bit for the block, main memory is not updated;
- on misses it updates the block in main memory not bringing that block to the cache;
- Subsequent writes to the same block, if the block originally caused a miss, will generate misses all the way and result in very inefficient execution.

The following applet shows the dynamics of interaction policies. By clicking on white squares of the applet one can chose the policy to test depending on whether it is read or write, hit or miss. On read hit there is not much choice of policies. The block is just read from cache to CPU.

The following applet will show you the dynamics of interaction policies.

For more hands-on approach the problem (with solution) is provided for you.