Computer Architecture
Homework 5
Theme: Cache Memory
All questions carry equal weight. You must show work to receive credit.

---

1. What is the function of a replacement algorithm? Do you need a replacement algorithm for direct mapping?

   Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced. For direct mapping, there is only one possible line for any particular block, and no choice is possible. For the associative and set-associative techniques, a replacement algorithm is needed.

2. Define the two write policies in about ten sentences total. Discuss the relative merits of the policies.

   When a block that is resident in the cache is to be replaced, there are two cases to consider.

   - If the old block in the cache has not been altered, then it may be overwritten with a new block without first writing out the old block.

   - If at least one write operation has been performed on a word in that line of the cache, then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block.

   Two write policies are possible – Write through and Write back.

   The simplest technique is called write through. Using this technique, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid. Any other processor–cache module can monitor traffic to main memory to maintain consistency within its own cache. The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck.

   An alternative technique, known as write back, minimizes memory writes. With write back, updates are made only in the cache. When an update occurs, a dirty bit, or use bit, associated with the line is set. Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set. The problem with write back is that portions of main memory are invalid, and hence accesses by I/O modules can be allowed only through the cache. This makes for complex circuitry and a potential bottleneck.

3. Describe a simple way of implementing an LRU replacement algorithm in a 4-way set associative cache.

   LRU replacement algorithm implementation for 4-way set associative cache can be as follows:

   - Use 2 bits per cache line in set to store the USE flag/setting. Hence, the valid values for USE flag/setting are 00, 01, 10, 11.
   - Upon access to a line, set its USE flag = 00; This marks that the line is accessed most recently; Then increment the USE flag/setting of all other cache lines in the set by 1. This marks that they grew older by one more access.
   - Repeat the above step for each cache hit.
   - Upon cache miss, a line has to be replaced from set to make room for a new block from memory.
   - From the above, we see that, the line with highest USE flag/setting is the least accessed line of all available lines in set. Hence, that line can be replaced with the newly read block from main memory. Then set its USE flag to 00 as this is the recent accessed one and increment the USE flag of all other cache lines in set by 1, as earlier.
   - Repeat the above step for each cache miss.

4. What is the impact of the design decision of line size on hit ratio? Discuss the various approaches of cache coherency in a multiprocessor system (in about 10 sentences). Discuss the relative merits of unified vs. split (code/data) cache design.

   Line size: **4 points**

   When a block of data is retrieved and placed in the cache, not only the desired word but also some number of adjacent words are retrieved. As the block size increases from very small to larger sizes, the hit ratio will at first increase because of the principle of locality, which states that data in the vicinity of a referenced word are likely to be referenced in the near future. As the block size increases, more useful data are brought into the cache. The hit ratio will begin to decrease, however, as the block becomes even bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced. Two specific effects come into play:

   - Larger blocks reduce the number of blocks that fit into a cache. Because each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched.
   - As a block becomes larger, each additional word is farther from the requested word and therefore less likely to be needed in the near future.

   Cache coherency: **5 points**

   In a bus organization in which more than one device (typically a processor) has a cache and main memory is shared, a new problem is introduced. If data in one cache are altered, this invalidates not only the corresponding word in main memory, but also that same word in other caches (if any other cache happens to have that same word). Even if a write-through policy is used, the other caches may contain invalid data. A system that

prevents this problem is said to maintain cache coherency. Possible approaches to cache coherency include the following:

- Bus watching with write through: Each cache controller monitors the address lines to detect write operations to memory by other bus masters. If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry. This strategy depends on the use of a write-through policy by all cache controllers.
- Hardware transparency: Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches. Thus, if one processor modifies a word in its cache, this update is written to main memory. In addition, any matching words in other caches are similarly updated.
- Non-cacheable memory: Only a portion of main memory is shared by more than one processor, and this is designated as non-cacheable. In such a system, all accesses to shared memory are cache misses, because the shared memory is never copied into the cache. The non-cacheable memory can be identified using chip-select logic or high-address bits.

Unified Vs Split Caches **4 points**

Earlier, a single cache is used to store references to both data and instructions. More recently, it has become common to split the cache into two: one dedicated to instructions and one dedicated to data. These two caches both exist at the same level, typically as two L1 caches. When the processor attempts to fetch an instruction from main memory, it first consults the instruction L1 cache, and when the processor attempts to fetch data from main memory, it first consults the data L1 cache.

There are two potential advantages of a unified cache:

- For a given cache size, a unified cache has a higher hit rate than split caches because it balances the load between instruction and data fetches automatically. That is, if an execution pattern involves many more instruction fetches than data fetches, then the cache will tend to fill up with instructions, and if an execution pattern involves relatively more data fetches, the opposite will occur.
- Only one cache needs to be designed and implemented.

Despite these advantages, the trend is toward split caches, particularly for superscalar machines such as the Pentium and PowerPC, which emphasize parallel instruction execution and the prefetching of predicted future instructions.

- The key advantage of the split cache design is that it eliminates contention for the cache between the instruction fetch/decode unit and the execution unit. This is important in any design that relies on the pipelining of instructions.

5. Define spatial and temporal locality. Consider the following code:

```
for (i=0; i < 50; i++)
      for (j =0; j < 30; j++)
            a[i] = a[i] * j;
```

   a. Give an example of spatial locality in the above code. Definition 4 points, example 3 points

      Spatial locality is observed when data in memory address locations closer to the current accessed location are accessed.

      Upon execution of the provided code segment, spatial locality will be observed as each element of the array is being accessed in sequence, i.e. from a[0]… until a[49]. Array elements are usually stored in consecutive memory locations.

      In the given code, spatial locality is due to the first for loop.

      for (i=0; i < 50; i++)

      In addition, spatial locality will be observed during program execution as program instructions will be referenced from sequential locations in memory.

   b. Give an example of temporal locality in the above code.

      Temporal locality is observed when same memory location is accessed several times over a short time period.

      Upon execution of the provided code segment, temporal locality will be observed as each element of the array 'a' is accessed 30 times as part of inner loop execution over a short period, before moving the control to operate the set of operations on next array element.

      In the given code, temporal locality is due to the second for loop.

      for (j =0; j < 30; j++)
            a[i] = a[i] * j;

6. Consider a memory system with the following parameters:

   Tc = *5ns*, Cc= $*$10^{-6}$/bit*

   Tm = *50ns*, Cm = $*$10^{-7}$/bit*

   a. What is the cost of 4 GB of main memory? 4 points

      Cost of 4 GB of main memory = $4 \times 10^9 \times 8$ bits $\times$ Cm
      $= 4 \times 10^9 \times 8$ bits $\times$ $\$10^{-7}$/bit = 3200$

   b. What is the cost of 4 GB of main memory using cache only technology? 4 points

      Cost of 4 GB of main memory using cache technology
      $= 4 \times 10^9 \times 8$ bits $\times$ Cc

$= 4 \times 10^9 \times 8$ bits $\times \$10^{-6}$/bit = 32000$

c. If the effective access time is 20% greater than cache access time, what is the hit ratio H?

Given, Tc = 5ns, Tm = 50ns

Given, effective access time is 20% greater than cache access time I.e. Te = 6ns

Average access time = hit ratio × Cache access time + miss ratio × (Cache access time + Memory access time) 1 points

Tavg (in ns)= HTc + (1-H) × (Tc+Tm) = 5H + (1-H)55 = 55-50H

6 = 55-50H 2 points

50H = 49

Hit ratio H=49/50 = 98% 2 points

7. A computer has cache, main memory and a disk used for virtual memory. If a referenced word is in the cache, 15 ns are required to access it. If it is in the main memory, but not in the cache 60 ns are needed to load into the cache, and then the reference is started again. If the word is not in the main memory, 10 ms are required to fetch the word from the disk, followed by 70 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.95 and the main memory hit ratio is 0.6. What is the average time in ns required to access a referenced word on this system?

Given,

- Cache hit ratio = 0.95,
- Main memory hit ratio = 0.6

Access times are as given below:

- for cache, Tc=15 ns; 2 points
- for main memory, Tm=60ns; 2 points
- for disk, Td=10ms=10000000ns 2 points

Average time to access a referenced word in system
= cache hit ratio × Tc + cache miss ratio × (memory hit ratio × (Tm+Tc) + memory miss ratio × (Td+Tm+Tc)) 4 points
= 0.95 × Tc + 0.05 × (0.6 × (Tm + Tc) + 0.4× (Td + Tm + Tc))
= 0.95 × 15 + 0.05 × (0.6 × 75 + 0.4 × 10000075)
= 200018 nsec 4 points