Cache Memory System

0.1 Introduction to cache

The Speed of CPU v/s Main memory is typically 10:1. To compensate for this gate, a high speed *Cache memory* (M_1) of relatively small size is provided between *main memory* (M_2) and CPU, forming a (M_1, M_2) hierarchy of memories. Some mechanism is provided in cache so that the access t_A of main memory is approximately equal to t_{A_1}, the access time of cache memory. So that the cache hit ratio is close unity. The *hit ratio* of cache memory is defined as for what percentage of the totals requests, the data/instructions were available in the cache memory. Hence, if total read requests made to cache were n and out of this for m requests, the data/instructions were already available in the cache, i.e., it did not require them to be fetched from RAM to meet this request, the hit ratio is,

$$\frac{m}{n}. \tag{1}$$

Note that hit ratio is less than unity, but is close to unity, almost 0.99 (99%) in most cases. This is possible because the Data and instructions are pre-fetched in cache. Due to the *locality of reference* (i.e. the next data or instructions are likely to be physically close to the earlier) nature of the programs other instructions/data are likely to be found cache. Cache is usually not visible to programmers. The approach is: One can search a desired address's word in a cache for match; and does this for

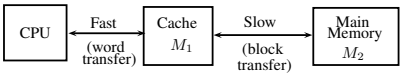


Figure 1: Memory Hierarchy.

all locations in cache simultaneously. Hence, it can check the availability of data in cache in single clock cycle.

Cache size should be small, so that the *average cost* is that of RAM. At the same time, the Cache should be large enough so that *average access time* is close to that of cache.

The VAX 11/780 system comprised of a 8-KB *general purpose cache*, Motorola 68020 has 256 byte on-chip *instruction cache*, the 8086 CPU has 4 / 6 byte instruction cache, the Intel Dual core processor has L_1 : 512KB-2MB, L_2 : 1-2MB. Instruction cache v/s General Purpose cache has the difference that, while in the first, only instructions can be cached, for the later the instructions and data both can be cached.

0.2 Cache Working Principle

When processor attempts to read a word, its presence is first checked in cache. If found, it is delivered to processor, else a block of fixed number of words is read into cache which contains this word also. Simultaneously, the word also is delivered to CPU. If the word was found in cache, it is *cache hit* else it is *cache miss*. The figure 10 shows the relative position of cache memory.

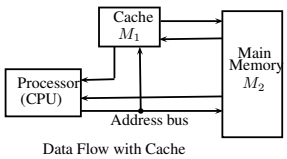


Figure 2: Cache position with respect to CPU and memory.

0.2.1 Cache Design

Let number of *words* in the RAM are 2^n, each addressable by an address of n-bits, RAM is of M *blocks*. Word size may be 8, 16, 32, 64 bits. Memory consists fixed length blocks, each of size K words$(= 2^w)$, and total blocks M in the RAM are $M = 2^n \div K$. Cache has size of $m = 2^r$ blocks or *lines*. Each line has 2^w words followed with a *tag field* and a *control-field*. The control field (also called dirty bit) is to show line has been modified. A *tag* identifies which particular blocks are presently in cache.

Mapping Function: Since lines in cache, $m \ll M$ (blocks in RAM), an algorithm is needed to map M to m. That determine what memory *block* occupies what cache *line*. Figure 3 shows the address mapping in cache, while the figure 4 shows flow-chart for accessing a word from cache.

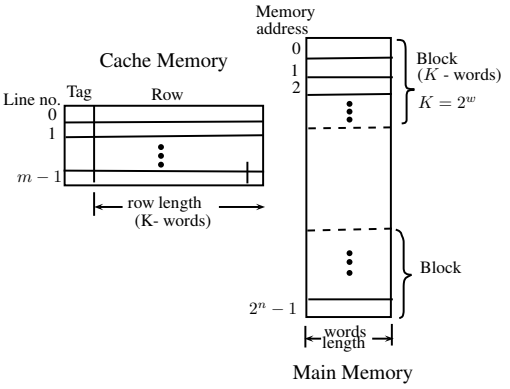


Figure 3: Address mapping.

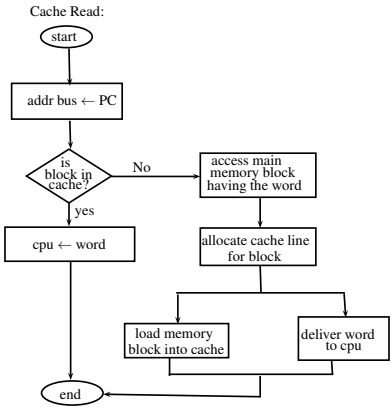


Figure 4: Flow-chart for Accessing a word.

There are three address mapping techniques, namely, *direct, associative,* and *set associative.*

0.3 Direct-Mapped Cache

In the *direct mapped* cache each block maps to one and only one line in cache and that is fixed always. The mapping is expressed by:

$$i = j \mod m$$

where, j is main memory block number, m is total number of lines in the cache, and i is cache line number to which block j of main memory is mapped.

Let the the words address space is n-bits. The RAM is divided into fixed length K blocks each of 2^w words. A word may be equal to a byte. The number of blocks in RAM $= 2^n/K = M = 2^s$ (here s is number of bits required to represent block a number in RAM).

Let us assume that, $m = 2^r$ is number of rows in the cache, each of size one block $= 2^w$ words. The r is number of bits required for a *line* number in the cache. The address space in RAM is illustrated in figure 5.

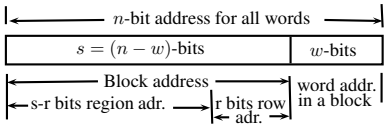


Figure 5: Address Space in RAM.

There is need of some kind of mapping between 2^r lines in cache v/s 2^s number of blocks in RAM. This is because only 2^r out of 2^s blocks can be resident in the cache. Note that a block fits exactly in a cache line.

The *Direct mapping* is carried out as follows. If $2^S \mod 2^r = 0$, then RAM is divided into $2^s \div 2^r = R$ Regions as follows:

Regions No.	Block Nos. in RAM	Mapped Cache lines
0	$0, \ldots, 2^r - 1$	$0, \ldots, 2^r - 1$
1	$2^r, \ldots, 2.2^r - 1$	$0, \ldots, 2^r - 1$
...	...	...
$R-1$	$R-1 \times 2^r \ldots R.2^r - 1$	$0, \ldots, 2^r - 1$

If 2^s is total number of blocks in RAM, and 2^r is number of lines in cache, with line size equal to block size, i.e., 2^w words, then i^{th} block of RAM is mapped to j^{th} line of cache, so that $j = i \ mod \ 2^r$. For example, line (also called row) 5 in cache will hold one block (5th block) from only one of the Regions out of $0, \ldots, R-1$. Thus, a row address corresponding

to r bits from address field is a direct index to the block position in a Region. The address mapping is illustrated in figure 6.

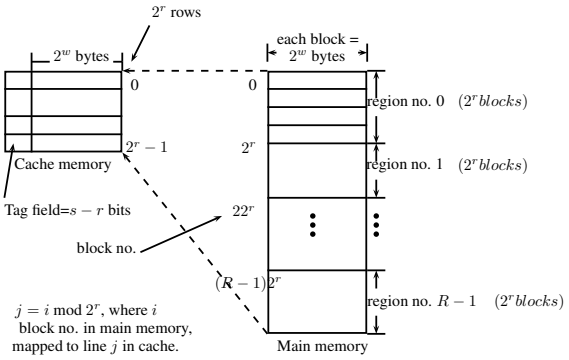
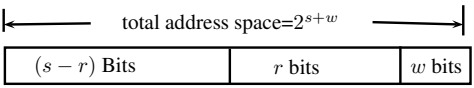


Figure 6: Direct Address mapping.

For a match, the r bits (i.e., index, having value, say, 1011) is taken from address field, and the *tag* field in row number 2^{1011} of cache is compared with the $s-r$ bits of address field. If they matched equal, then it is *cache hit*, else *cache-miss*. The figure 7 shows the address mapping.



The r bits acts only as index to a row in cache.
If the tag field of cache macthes with $s - r$ bits of addr space for an index, it is hit.
It requires only one comparision per address in the direct mapped cache.

Figure 7: Address mapping.

We note that only one comparison is required to test for a hit. Hence, the circuitry required for implementation is simple.

By closely observing this process of direct mapped cache we note a serious limitation. For example, if there is frequent reference to same block number from different Regions, it will require too many swapping resulting the over all processing to be slow. It is called **thrashing**. Imagine that two same block number are required from two different Regions, say block number 5, it would not be possible to accommodate them at the same time in the cache. This is because in the direct mapped cache all the same number blocks from a Region are mapped to the same line of cache.

Example 0.3.1 *Direct Mapped Cache.*

Let $n = 32$, that is, $4GB$ size, $w = 4$. One block size is $K = 2^4 = 16$ bytes. Number of total blocks $= 2^{32-4} = 2^{28} = 256$ million. Size of cache $= 4MB$, $m = 4MB$, number of lines in cache = 4MB/16 bytes$=2^{22}/2^4 = 2^{18}$. Total no. of Regions $R =$ (total blocks)/(blocks in cache)$= 2^{28}/2^{18} = 2^{10}$. Hence, $n = 32, w = 4, r = 18, s = n - w = 32 - 4 = 28$. Tag size in cache is $s - r = 28 - 18 = 10$.

Consider the address of RAM location, in binary format, indicated along with the bit positions, 0 to 31.

——————Address bits values——————
1 0 0 0.1 1 1 1.1 0 0 1.0 1 1 0.1 1 0 0.1 0 0 1.1 0 1 0.0 0 0 1
1 0 9 8.7 6 5 4.3 2 1 0.9 8 7 6.5 4 3 2.1 0 9 8.7 6 5 4.3 2 1 0
——————Address positions A31-A0——————-

We note that the word address in a block is $(A_3 - A_0) = 0001 = 1$, line address $(A_{21} \ldots A_4) = 2^r - 1 = 01\ 0110\ 1100\ 1001\ 1010_2$, Region address $= A_{31} \ldots A_{22} = 10$ tag bits. The line address is common for all the 2^{10} Regions. Thus, if tag bits in the cache for line number $01\ 0110\ 1100\ 1001\ 1010_2$ matches with the Region number address $A_{31} \ldots A_{22}$, then this block $(A_4 - A_{21}) = 01\ 0110\ 1100\ 1001\ 1010_2$, is in cache, else not. The line number in the cache is expressed by $i = j \ mod \ m = (A_{31} - A_4) \ mod \ 2^{18} = 1\ 0\ 0\ 0.1\ 1\ 1\ 1.1\ 0\ 0\ 1.0\ 1\ 1\ 0.1\ 1\ 0\ 0.1\ 0\ 0\ 1.1\ 0\ 1\ 0 \ mod \ 2^{18} = 1\ 0\ 0\ 0.1\ 1\ 1\ 1.1\ 0$. (The decimal point is to split into lots of fours, and not a real decimal).

0.3.1 Block Replacement Algorithm

The Objective of block replacement algorithm is that every request for instruction/data fetch from cache must be met by the cache. Blocks transfer to cache will improve the hit. *Hit ratio* = (total hits)/(total attempts), which should be close to unity (normally > 0.99).

The objective is partly achieved by algorithm used for block replacement. What happens when request for loading new block in cache does not find space in cache? One of the row (block frame or line) is to be moved back to main memory to create space for new block. The question arises, *Which block should be removed?* An approach can be, the one which had remained in the cache for a longest time, called *FIFO first in first out* algorithm. The approach is to transfer that block which was used for a long times: *LRU* (least recently used) algorithm, or the one which is not frequently used: *least frequently used* algorithm. In *Random*

time: algorithm, a random number is generated, and a block is selected correspondingly for removal, i.e., a *random block selected.*

A block should be written back to main memory only if it is changed, otherwise it can be over written in cache as it identical copy exists there. For each row in cache (it corresponds to a block) a special bit, called, *sticky bit or dirty bit*, is allocated. If the block is modified, this bit is *set* else it remain *reset.* This bit helps to decide whether to write this line back to RAM or not.

0.4 Associative Cache Memories

Many storage and retrieval problems require accessing certain subfields within a set of records, *what is John's ID no. and age?*

```
Name    ID number Age
--------------------
J. Smith 124      24
J. Bond  007      40
A. John  106      50
R. Roe   002      19
J. Doe   009      28
--------------------
```

The row 3 has no logical relationship to Jones? To find Jones it requires to search the entire table using name sub-field as an address. But, that would be too slow. The associative memory simultaneously examines all the entries. They are known as *content addressable memories.* The subfield chosen is called *tag* or *key.* Items stored in Associative memories can be viewed as: KEY, DATA pairs.

Associative mapping Cache Memories: A memory block $(0, \ldots, 2^s - 1)$ of main memory can go (mapped to) anywhere (any line) in cache. This removes the drawback of *Direct mapped* cache. A Cache of size 2^r lines (r bits address) may hold any 2^r blocks out of total 2^s blocks of RAM. Larger is size of cache, the larger number of RAM blocks it can hold at any time. These cache memories are called fully associative cache memories.

Since any 2^r blocks out of total 2^s blocks of RAM may exists in cache, we need a *tag field of s bits* along with each line of cache. Total address space is $n = s + w$ bits, there are total 2^s blocks in RAM, each block having 2^w words. Following are steps to find out if a block number $b = 2^x$ is in cache:

1. s bits from address space are compared (**in parallel**) with all the tags of 2^r lines of cache.

2. The line in which match occurs, it holds the block having required word corresponding to the address field.

3. If no match occurs with any tag of cache, it is a cache-miss, and the required block is fetched from RAM.

4. Otherwise, it is cache miss and and the block having the required word will be loaded into the cache.

0.5 Set Associative Cache

If there are ten Cache lines to which a RAM location are mapped, ten cache entries must be searched. It takes power, chip area, and time. Caches with more associativity has fewer misses. The rule of thumb is "Doubling the associativity, from direct mapped to 2-way, or from 2-way to 4-way, has same effect on hit ratio as doubling cache."

A set-associative scheme is a hybrid between fully associative and direct mapped, and a reasonable compromise between complex hardware (of fully associative) v/s simple of direct-mapped. The solution is *set associative cache.*

Consider a cache of size 8 lines, and RAM of 32 blocks. If the cache is fully associative, the block (say 12 number) can go into any of the 8 rows of cache.

In a set associative cache, the block 12 will be mapped to a set in the cache. Set can be of size 1, 2, 4, 8 rows now. For two sets (each of 4-rows), block 12 will be mapped to a set, i.e., $\{1, 2\}$. Thus any block x of RAM can be mapped into any of the 4 lines of a set. This is called *4-way set associative cache.*

If set is represented by u-bits in address field, then set number can be found by index of u bits of address. The tag field of each row $= s - u$ bits.

Compromises: It requires complex comparator hardware (to find the correct slot) out of a small set of slots, instead of all the slots. Such hardware is linear in the number of slots (row in a set).

However, it provides flexibility of allowing up to N cache lines per slot for an N-way set associative. Figure 8 shows the addressing details for set associative cache memory system.

Algorithm to find cache hit:

1. Pick up the u bits out of total $(s-u)+u$ of block address; use the u bits as index to reach to 2^u th set in the cache.

2. compare(*in parallel*) the $s-u$ bits from address field with tag fields of all the 2^{s-u} lines in that set.

3. If any match occurs, it is hit, and line whose tag is matched, has the required block. So the byte from that word is transferred to CPU. Else, it is miss, and the block is replaced from RAM.

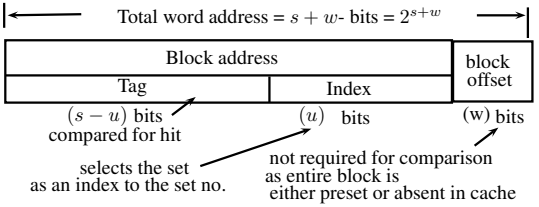


Figure 8: Set Associative Cache memory addressing details.

0.6 Analysis

In fully associative, length of u is zero. For a given size of cache, we observe following:

- tag-index boundary moves to right

  $\Rightarrow$ decrease index size u

  $\Rightarrow$ increase number of blocks per set

  $\Rightarrow$ increase size of tag field

  $\Rightarrow$ Increase associativity

- Direct mapped cache is set associative with set size =1.

- tag-index boundary moves to left

  $\Rightarrow$ increases index size u

$\Rightarrow$ decreases number of blocks per set

$\Rightarrow$ decreases in size of tag field

$\Rightarrow$ Decrease in associativity

0.7 Write policy

When new block is to be fetched into cache, and there is no space in cache to load that block, there is need to create space in cache. For this one of the old blocks needs to be transferred to main memory. What block is to considered for this is decided as per the algorithms used.

If a line or block frame is not modified, then it may be over written, as it does not require to update into RAM. If the block is modified in RAM (possible when multiple processors with multiple caches exist or it is modified by device) then cache is invalid.

One of the technique to solve this problem is called *write through*, which ensures that main memory is always valid. Consequently if any change has taken place in cache line, it should be updated into RAM immediately. The disadvantage of this method is wastage of CPU-bandwidth due to frequent updating of RAM. The advantage is, easier to implement, cache is always clean, read misses (in cache) never lead write to RAM.

When write through takes place, the *cpu stalls*. To solve this, the data is written to buffer, before it goes to RAM.

write back: The cache is written (updated) into RAM only when block is to be removed from cache to create a space in cache and the dirty bit of the cache line is found set. This approach consumes lesser cpu-memory BW for writes. However, it is suitable for servers (with multiple processors). The approach is also good for embedded applications as it saves power.

Shared memory Architecture: There are two main problems with shared memory system: performance degradation due to contention, and coherence problems. Multiple processors try to access the shared memory simultaneously. Having multiple copies of data, spread throughout the caches, might lead to a coherence problem. The copies in the caches are *coherent* if they are all equal to the same value.

0.8 Cache coherence

The dictionary meaning of *Coherence means*, a *sticking* or *cleaving* together; union of parts of the same body; cohesion. Connection or dependence, proceeding from the subordination of the parts of a thing to one principle or purpose, as in the parts of a discourse, or of a system of philosophy; a logical and orderly and consistent relation of parts; consecutiveness.

Coherence of discourse, and a direct tendency of all the parts of it to the argument in hand, are most eminently to be found in him. –Locke.

Figure 9 shows multiple caches attached to a single RAM memory used as shared resource. The sequence of operations may be: Client 1 modifies the cache 1, but does not write to main memory. This is followed with, client 2 accesses that memory block from RAM which was modified in the cache 1. The result is invalid data access by client 2. This is due to lack of coherence.

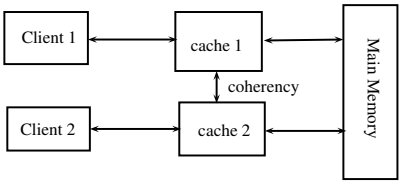


Figure 9: Multiple caches using a shared resource.

0.8.1 Achieving coherence

Principle of coherence: See the figure 9. The client 1 reads location x, modifies it, and writes location x. In between no other processor accesses x. This results to memory *consistency*, else not.

Following are the mechanisms for Cache coherence.

Directory-based coherence

Data being shared is placed in a common directory, which acts as a filter through which processor must ask permission to load an entry from the RAM to cache. When entry is changed, the directory updates or invalidates other caches with that entry.

Snoopy-bit

Individual cache monitors address lines of RAM that have been cached. When write operation is observed in a location x whose copy is within cache, then cache controller invalidates its own copy of that location x.

Snarfing

Cache controller watches both address and data lines, and if RAM location x gets modified, the cache updates its own copy of x.

The coherence algorithms above, and others maintains consistency between all caches in a system of distributed shared memory.

0.9 Snooping Protocols

They are based on watching bus activities and carry out the appropriate coherency commands when necessary. Each block has a state associated with it, which determines what happens to the entire contents of the block. The state of a block might change due to: *Read-Miss, Read-Hit, Write-Miss, and Write-Hit.*

Cache miss: Requested block is not in the cache or it is in the cache but has been invalidated.

Snooping protocols differ in whether they *update or invalidate* shared copies in remote caches in case of a write operation. They also differ as from where to obtain the new data in the case of a cache miss.

The State of the cache may be *Valid*, *Invalid.*The events which may result to these states are: *Read-Hit*, *Read-Miss*, *Write-Hit*, *Write-Miss*, *Block replacement.*

In the event of *Write-Invalidate and Write-Through* the memory is always consistent with the most recently updated cache copy. The other approach of update is *Write-Invalidate and Write-Back.*

0.10 Instruction Cache, data cache, Cache levels

The *Harvard Cache:* It Separate data and instruction caches. Allows accesses to be less random. Both have locality of reference property.

Multilevel cache hierarchy: Internal cache, external cache, L_1, L_2, L_3, etc.

Cache Performance: The cache performance is determined by *Hit Ratio* and *Average memory access time*. The *Average memory access time* is defined as follows:

Let t_h is Hit time, t_{mr} is miss rate, and p_m is miss penalty (i.e., number of clocks missed), then *average memory access time* is:

$$t_A = t_h + t_{mr} \times p_m \tag{2}$$

Consider that t_{cc} is number of CPU execution cycles for a given program, t_{msc} is memory stall cycles, and t_c is one clock cycle time. Then, *CPU time* t_{cpu} for execution of this program is,

$$t_{cpu} = (t_{cc} + t_{msc}) \times t_c \tag{3}$$

seconds.

Consider that memory stall cycles are t_{msc} in n instructions, Then average *miss penalty* p is:

$$p = \frac{t_{msc}}{n} \tag{4}$$

Which is difficult to calculate. So, we adopt other approach. Let total misses are m, number of instruction during that time are n, *total miss latency* is t_{tml}, and *overlapped miss latency* is t_{oml}, the net miss latency is:

$$\frac{m}{n} \times (t_{tml} - t_{oml}) \tag{5}$$

The Overlapped miss latency is due to out-of-order CPUs stretching the hit time.

0.11 Cache Positioning

A cache is always positioned between RAM and CPU. A programmer is unaware of its position, because the most part of cache operation and management is done by hardware and a very small by operating system. Figure 10 shows the cache Positioning.

0.11.1 Virtual to Physical Address Translation

In virtual memory system, the logical address needs to be translated into physical address, before instruction/data is fetched.

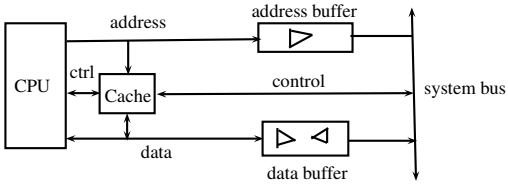


Figure 10: Cache positioning.

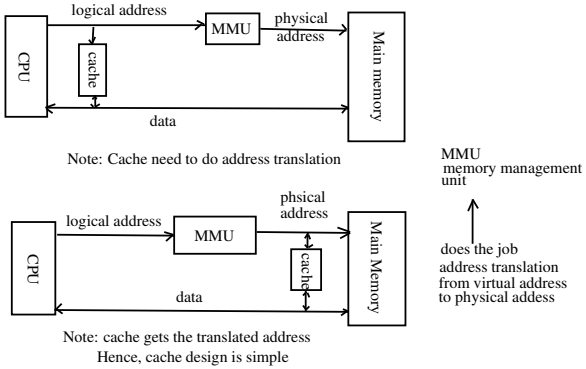


Figure 11: Cache positioning.

For cache memory this may be done by cache itself (in that case cache shall be fast) or by MMU (the cache will have to wait till the address translation is done).(see figure 11).

In figure 10, cache itself is doing address translation, in second (figure 11) address translation is done by MMU and cache see only the physical address.

Exercises

1. Self test questions Direct mapped cache:

   (a) If two same block numbers from two Regions are required in the cache, what is consequence? That is, can they be present same time in cache?

   (b) Does the CPU fetch directly from RAM, if cache is full?

   (c) A block can be resident at how many alternate locations in cache?

2. For set associative cache, What are the effect of:

   (a) Index boundary max left?

   (b) Index boundary max right?

   (c) How far it can go left/right?

   (d) What is best position of index boundary?

3. A block-set-associative cache consists of a total 64-blocks divided into 4-block sets. The main memory consists of 4096 blocks, each consisting of 128 words.

   (a) How many bits are there in a main memory address?

   (b) How many bits are there in each of the TAG, SET, and WORD fields?

4. A 2-way set associative cache memory uses blocks of four words. The cache can accommodate a total of 2048 words from main memory. The main memory size if 128K $\times$ 32.

   (a) Formulate all the required information to construct the cache memory.

   (b) What is size of the cache memory?

5. Let us consider a memory hierarchy (main memory + cache) given by:

   - Memory size 1 Giga words of 16 bit (word addressed)
   - Cache size 1 Mega words of 16 bit (word addressed)
   - Cache block size 256 words of 16 bit

   Let us consider the following cache structures:

   - direct mapped cache;
   - fully associative cache;
   - 2-way set-associative cache;
   - 4-way set-associative cache;
   - 8-way set-associative cache.

   (a) Calculate the structure of the addresses for the previous cache structures;

   (b) Calculate the number of blocks for the previous cache structures;

(c) Calculate the number of sets for the previous set associative caches.

6. A cache memory is usually divided into lines. Assume that a computer has memory of 16 MB, and a cache size of 64 KB. A cache block can contain 16 bytes.

   (a) Determine the length of the tag, index and offset bits of the address for: a. Direct Mapped Cache, b. 2-way set Associative Cache, c. Fully Associative Cache

   (b) Assuming a memory has 32 blocks and a cache consists of 8 blocks. Determine where the 13th memory block will be found in the cache for: a. Direct Mapped Cache, b. 2-way Set Associative Cache, c. Fully Associative Cache

7. Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

   (a) How is the 16-bit memory address divided into tag, line number, and byte number?

   (b) Into what line would bytes with each of the following addresses be stored?
   0001 0001 1001 1011
   1100 0011 0010 0100
   1010 1010 0110 1010

   (c) Suppose the byte with address *0001 0001 1001 1011* is stored in the cache.What are the addresses of the other bytes stored along with this?

   (d) How many total bytes of memory can be stored in the cache?

   (e) Why is the tag also stored in the cache?

8. In a cache-based memory system using FIFO for cache page replacement, it is found that the cache hit ratio H is unacceptedly low. The following proposals are made for increasing H:

   (a) Increase the cache page size.

   (b) Increase the cache storage capacity.

   (c) Increase main memory capacity.

   (d) Replace the FIFO policy by LRU.

   Analyse each proposal to determine its probable impact on H.

9. Consider a system containing a 128-byte cache. Suppose that set-associative mapping is used in the cache, and that there are four sets each containing four lines. The physical address size is 32-bits, and the smallest addressable unit is the byte.

   (a) Draw a diagram showing the organization of the cache and indicating how physical addresses are related to cache addresses.

   (b) To what lines of the cache can the address $000010AF_{16}$ be assigned?

   (c) If the addresses $000010A_{16}$ and $FFFF7Axy_{16}$ are simultaneously assigned to the same cache set, what values can the address digits x and y have?

10. Discuss briefly the advantages and disadvantages of the following cache designs which have been proposed and in some cases implemented. Identify three nontrivial advantages or disadvantages (one or two of each) for each part of the problem:

    (a) *An instruction cache*, which only stores program code but not data.

    (b) A *two-level cache*, where the cache system forms a two level-memory hierarchy by itself. Assume that the entire cache subsystem will be built into the CPU.

11. A set associative cache comprises 64 lines, divided into four-line sets. The main memory contains 8K block of 64 words each. Show the format of main memory addresses.

12. A two-way set associative cache has lines of 16 bytes and a total size of 8 kbytes. The 64-Mbyte main memory is byte addressable. Show the format of main memory addresses.

13. Consider a 32-bit microprocessor that has an on-chip 16-Kbyte four-way set associative cache. Assume that the cache has a line size of four 32-bit words. Draw a block diagram of this cache showing its organization and how the different address field are used too determine a cache hit/miss. Where in the cache is the word from memory location $ABCDE8F8_{16}$ mapped?

Bibliography

[1] John P. Hayes, "Computer Architecture and Organization", 2nd Edition, McGraw-Hill, 1988.

[2] William Stalling, "Computer Organization and Architecture", 8th Edition, Pearson, 2010.

[3] M. Morris Mano, "Computer System Architecture", 3rd Edition, Pearson Education, 2006.

[4] Carl Hamacher, Zvono Vranesic, and Safwat Zaky, "Computer Organization ", , 5th edition, McGrawhill Education, 2011. (chapter 7)