**1.** Caches are important to providing a high-performance memory hierarchy to

processors. Below is a list of 64-bit memory address references, given as word addresses.

0x03, 0xb4, 0x2b, 0x02, 0xbf, 0x58, 0xbe, 0x0e, 0xb5, 0x2c, 0xba, 0xfd

*1.1* For each of these references, identify the binary word address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list whether each reference is a hit or a miss, assuming the cache is initially empty.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hex Memory Reference | Binary Reference | Tag | Index | Hit / miss |
| 0x03 | 0000 0011 | 0 | 3 | M |
| 0xb4 | 1011 0100 | b | 4 | M |
| 0x2b | 0010 1011 | 2 | b | M |
| 0x02 | 0000 0010 | 0 | 2 | M |
| 0xbf | 1011 1111 | b | f | M |
| 0x58 | 0101 1000 | 5 | 8 | M |
| 0xbe | 1011 1110 | b | e | M |
| 0x0e | 0000 1110 | 0 | e | M |
| 0xb5 | 1011 0101 | b | 5 | M |
| 0x2c | 0010 1100 | 2 | c | M |
| 0xba | 1011 1010 | b | a | M |
| 0xfd | 1111 1101 | f | d | M |

*1.2* For each of these references, identify the binary word address, the tag, the index, and the offset given a direct-mapped cache with two-word blocks and a total size of eight blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hex Memory Reference | Binary Reference | Tag | Cache Index | Block Offset | Hit / miss |
| 0x03 | 0000 0011 | 0 | 1 | 1 | M |
| 0xb4 | 1011 0100 | b | 2 | 0 | M |
| 0x2b | 0010 1011 | 2 | 5 | 1 | M |
| 0x02 | 0000 0010 | 0 | 1 | 0 | H |
| 0xbf | 1011 1111 | b | 7 | 1 | M |
| 0x58 | 0101 1000 | 5 | 4 | 0 | M |
| 0xbe | 1011 1110 | b | 7 | 0 | H |
| 0x0e | 0000 1110 | 0 | 7 | 0 | M |
| 0xb5 | 1011 0101 | b | 2 | 1 | H |
| 0x2c | 0010 1100 | 2 | 6 | 0 | M |
| 0xba | 1011 1010 | b | 5 | 0 | M |
| 0xfd | 1111 1101 | f | 6 | 1 | M |

*1.3* You are asked to optimize a cache design for the given references. There are three

direct-mapped cache designs possible, all with a total of eight words of data:

C1 has 1-word blocks,

C2 has 2-word blocks, and

C3 has 4-word blocks.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Word Address | Binary Address | Tag (5 bits in hex) | Index 1, block size = 1 word (3 bits) | Hit/Miss | Index 2, block size = 2 word (2 bits) | Hit/Miss | Index 3, block size = 4 word (1 bit) |  Hit/Miss |
| 0x03 | 0 0000 011 | 0x00 | 3 |M | 1 |M | 0 |M |
| 0xb4 | 1 0110 100 | 0x16 | 4 |M | 2 |M | 1 |M |
| 0x2b | 0 0101 011 | 0x05 | 3 |M | 1 |M | 0 |M |
| 0x02 | 0 0000 010 | 0x00 | 2 |M | 1 |M | 0 |M |
| 0xbf | 1 0111 111 | 0x17 | 7 |M | 3 |M | 1 |M |
| 0x58 | 0 1011 000 | 0x0b | 0 |M | 0 |M | 0 |M |
| 0xbe | 1 0111 110 | 0x17 | 6 |M | 3 |H | 1 |H |
| 0x0e | 0 0001 110 | 0x01 | 6 |M | 3 |M | 1 |M |
| 0xb5 | 1 0110 101 | 0x16 | 5 |M | 2 |H | 1 |M |
| 0x2c | 0 0101 100 | 0x05 | 4 |M | 2 |M | 1 |M |
| 0xba | 1 0111 010 | 0x17 | 2 |M | 1 |M | 0 |M |
| 0xfd | 1 1111 101 | 0x1F | 5 |M | 2 |M | 1 |M |

The miss rate of Cache 1 is 100%

The miss rate of Cache 2 is 83%

The miss rate of Cache 3 is 92%

Thus, Cache 2 is better with the lower miss rate since the Block size of cache 3 that becomes comparable to the Cache size(8 Words).

The competition for Blocks increases with the entire Block replaced if a single entry misses.

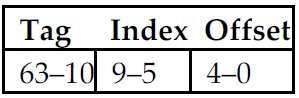
**2.** *Section 5.3* shows the typical method to index a direct-mapped cache, specifically

(Block address) modulo (Number of blocks in the cache). Assuming a 64-bit address and 1024 blocks in the cache, consider a different indexing function, specifically (Block address[63:54] XOR Block address[53:44]). Is it possible to use this to index a directmapped cache? If so, explain why and discuss any changes that might need to be made to the cache. If it is not possible, explain why.

Yes, it is. It satisfies the requirement that each unique set of bits in [63:54] there is exactly only 1 result of the XOR function for each of the 1024 combinations in [53:44] satisfying the unique cache index for any given memory address vector of 64 bits.

**3.** For a direct-mapped cache design with a 64-bit address, the following bits of the

address are used to access the cache.



*3.1* What is the cache block size (in words)?

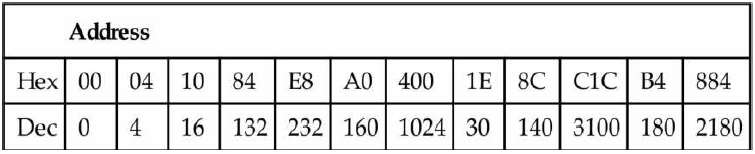
The Offset is 5 bit wide. Thus the block size is 32 bytes, 4 (64-bit) words.

*3.2* How many blocks does the cache have?

32blocks

*3.3* What is the ratio between total bits required for such a cache implementation over the data storage bits?

*Beginning from power on, the following byte-addressed cache references are recorded.*



The cache stores 32 Blocks\*4 Words/Block\*8 Bytes/word=1024 Bytes = 8192 bits

Total bits required: 8192 + 54\*32 + 1\*32 = 9952 bits

So the ratio: 9952/8192=1.2

*3.4* For each reference, list (1) its tag, index, and offset, (2) whether it is a hit or a miss, and (3) which bytes were replaced (if any).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Hex Memory Reference | Binary Reference | Tag | Index | Offset | Hit/Miss | Line Replaced |
| 0x00 | 0000 0000 | 00 | 0 0000 | 0 0000 | M | No |
| 0x04 | 0000 0100 | 00 | 0 0000 | 0 0100 | H | No |
| 0x10 | 0001 0000 | 00 | 0 0000 | 1 0000 | H | No |
| 0x84 | 1000 0100 | 00 | 0 0100 | 0 0100 | M | No |
| 0xe8 | 1110 1000 | 00 | 0 0111 | 0 1000 | M | No |
| 0xa0 | 1010 0000 | 00 | 0 0101 | 0 0000 | M | No |
| 0x400 | 0100 0000 0000 | 01 | 0 0000 | 0 0000 | M | Yes |
| 0x1e | 0001 1110 | 00 | 0 0000 | 1 1110 | M | Yes |
| 0x8c | 1000 1100 | 00 | 00100 | 01100 | H | No |
| 0xc1c | 1100 0001 1100 | 11 | 0 0000 | 1 1101 | M | Yes |
| 0xb4 | 1011 0100 | 00 | 0 0101 | 0 0100 | H | No |
| 0x884 | 1000 1000 0100 | 10 | 0 0100 | 0 0100 | M | Yes |

*3.5* What is the hit ratio?

The ratio : 4/12=33.333%

*3.6* List the final state of the cache, with each valid entry represented as a record of <*index, tag, data*>. For example,

**<0, 3, Mem[0xC00]-Mem[0xC1F]>**

<0, 3, Mem[0xC00]-Mem[0xC1F]>

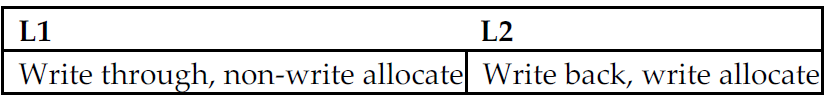
<4, 2, Mem[0x880]-Mem[0x89f]>

<5, 0, Mem[0x0A0]-Mem[0x0Bf]>

<7, 0, Mem[0x0e0]-Mem[0x0ff]>

**4.** Recall that we have two write policies and two write allocation policies, and their

combinations can be implemented either in L1 or L2 cache. Assume the following choices for L1 and L2 caches:



*4.1* Buffers are employed between different levels of memory hierarchy to reduce access latency. For this given configuration, list the possible buffers needed between L1 and L2 caches, as well as L2 cache and memory.

When the L2 cache has a high write miss penalty, the L1 cache will have a low write miss penalty because the latency between L1 and L2 is much lower than the latency between RAM and L2.A write buffer between L1 and L2 cache would pipeline the write to the L2 cache efficiently, which enables it to require only one cycle for the Write. Because of the buffer that holds data to be written into L2 from L1 and can be designed to be deep enough, it could stop stalls from subsequently writing misses in L1.Also, when replacing a dirty block in L2, L2 cache would benefit from write buffers between L1 and L2. The new block would be read into and held by the buffer between L1 and L2 before the dirty block is written to memory, only after which it could be overwritten in the L2 by the new block.

*4.2* Describe the procedure of handling an L1 write-miss, considering the components

involved and the possibility of replacing a dirty block.

When L1 write miss, the word will be written into L2 directly without writing back into L1 cache. If the results in L2 miss, it will be brought back from main memory.

Replacing a dirty block is possible. To make it happen, the dirty block must be written to memory firstly.

*4.3* For a multilevel exclusive cache configuration (a block can only reside in one of the L1 and L2 caches), describe the procedures of handling an L1 write-miss and an L1 readmiss, considering the components involved and the possibility of replacing a dirty block.

After an L1 cache write miss, the block will reside in L2 cache which is not in L1.At the same time, a read miss on the same block will require the block in L2 to be written back into memory, which transferred to L1 and invalidated in L2.

**5.** Consider the following program and cache behaviors.

page5image9502144

*5.1* Suppose a CPU with a write-through, writeallocate cache achieves a CPI of 2. What are the read and write bandwidths (measured by bytes per cycle) between RAM and the cache? (Assume each miss generates a request for one block.)

Instruction bandwidth:

Cycle per instruction: 2

0.5 instruction per cycle

Thus, 0.5\*0.30%\*64=0.096 bytes/cycle of data reads

Read Data bandwidth:

0.5\*(250/1000)\*2%\*64=0.16 bytes/cycle

Write Data bandwidth:

0.5\*(100/1000)\*2%\*64 = 0.064 bytes/cycle

Total read bandwidth:

0.096+0.16+0.064=0.32 bytes/cycle

Total write bandwidth:

0.5\*0.1\*0.8=0.04 bytes/cycle

*5.2* For a write-back, write-allocate cache, assuming 30% of replaced data cache blocks are dirty, what are the read and write bandwidths needed for a CPI of 2?

0.5\*(250+100)/1000\*2%\*30%\*64 = 0.0672 bytes/cycle

**6.** Media applications that play audio or video files are part of a class of workloads called “streaming” workloads (i.e., they bring in large amounts of data but do not reuse much of it). Consider a video streaming workload that accesses a 512 KiB working set sequentially with the following word address stream:

**0, 1, 2, 3, 4, 5, 6, 7, 8, 9 …**

*6.1* Assume a 64 KiB direct-mapped cache with a 32-byte block. What is the miss rate for the address stream above? How is this miss rate sensitive to the size of the cache or the working set? How would you categorize the misses this workload is experiencing, based on the 3C model?

The addresses stream are word addresses and the blocks are 32-byte (4 words).Thus, the miss rate will be 1/4 = 25%. The miss rate is not sensitive to the size of the cache or the working set. But the miss rate is sensitive to both the access pattern and the block size.

*6.2* Re-compute the miss rate when the cache block size is 16 bytes, 64 bytes, and 128

bytes. What kind of locality is this workload exploiting?

1/2=50%

1/8=12.5%

1/16=6.25%

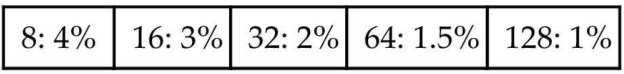
*6.3* “*Prefetching*” is a technique that leverages predictable address patterns to

speculatively bring in additional cache blocks when a particular cache block is accessed. One example of prefetching is a stream buffer that prefetches sequentially adjacent cache blocks into a separate buffer when a particular cache block is brought in. If the data are found in the prefetch buffer, it is considered as a hit, moved into the cache, and the next cache block is prefetched. Assume a two-entry stream buffer; and, assume that the cache latency is such that a cache block can be loaded before the computation on the previous cache block is completed. What is the miss rate for the address stream above?

In this case, after the first access, every access is a hit.

Therefore, the miss rate is almost 0 because 1/(2^19)-->0.

1. Cache block size (B) can affect both miss rate and miss latency. Assuming a machine with a base CPI of 1, and an average of 1.35 references (both instruction and data) per instruction, find the block size that minimizes the total miss latency given the following miss rates for various block sizes.



*7.1* What is the optimal block size for a miss latency of 20 ×B cycles?

8: 0.04\*20\*8 = 6.4

16: 0.03\*20\*16 = 9.6

32: 0.02\*20\*32 = 12.8

64: 0.15\*20\*64 = 19.2

128: 0.1\*20\*128 = 25.6

From the above, B = 8 is the best block size.

*7.2* What is the optimal block size for a miss latency of 24 +B cycles?

8: 0.04\*(24+8) = 1.28

16: 0.03\*(24+16) = 1.20

32: 0.02\*(24+32) = 1.12

64: 0.15\*(24+64) = 1.32

128: 0.1\*(24+128) = 1.52

From the above, B = 32 is the best block size.

*7.3* For constant miss latency, what is the optimal block size?

B = 128 is optimal. Because the miss rate will decrease while the block size is increasing. Thus, with a constant miss latency, a lower miss rate would be a better one.

**8.** In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that

main memory accesses take 70 ns and that 36% of all instructions access data memory. The following table shows data for L1 caches attached to each of two processors, P1 and P2.



*8.1* Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

P1: 1.515 GHz,

P2: 1.11 GHz

*8.2* What is the Average Memory Access Time for P1 and P2 (in cycles)?

P1:

70/0.66 = 107 cycles

AMAT = 1+0.08\*107 = 9.56 cycles

P2:

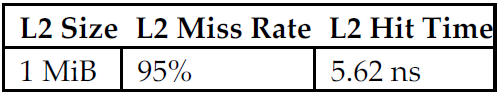
70/0.9 = 78 cycles

AMAT = 1+0.06\*78 = 5.68 cycles

*8.3* Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 and P2? Which processor is faster? (When we say a “base CPI of 1.0”, we mean that instructions complete in one cycle, unless either the instruction access or the data access causes a cache miss.)

*we will now consider the addition of an L2 cache to P1 (to presumably make up for its*

*limited L1 cache capacity). Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.*



P1:

1+0.08\*107+0.08\*0.36\*107=12.64 cycles

12.64\*0.66 = 8.34 ns

P2:

1+0.06\*78+0.06\*0.36\*78=7.36 cycles

7.36\*0.9 = 6.63 ns

From the above, P2 is faster.

*8.4* What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

L2 penalty = 5.62/0.66 = 9 cycles

AMAT = 1+0.08\*(9+0.95\*107) = 9.85 cycles>9.56 cycles

Thus, it is worse with L2 cache.

*8.5* Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 with the addition of an L2 cache?

1+0.08\*(9+0.95\*107) +0.08\*0.36\*(9+0.95\*107) =13.04 cycles

13.04\*0.66 = 8.60 ns

*8.6* What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P1 without an L2 cache?

Assume that L2 miss rate is x, then we will have:

1+0.08\*(9+x\*107) <= 9.56

Thus, X<=91.5%

Therefore, the miss rate should be less than 91.5%

*8.7* What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P2 without an L2 cache?

Assume that L2 miss rate is x, then we will have:

0.66[1+0.08\*1.36\*(9+x\*107)] <= 6.63 ns

Thus, X <= 69.29%

Therefore, the miss rate should be less than 69.3%

**9.** This exercise examines the effect of different cache designs, specifically comparing

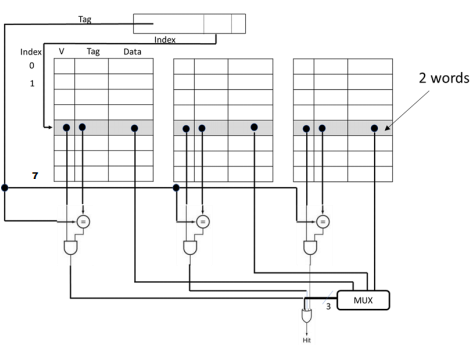
associative caches to the direct-mapped caches from *Section 5.4*. For these exercises, refer

to the sequence of word address shown below.

**0x03, 0xb4, 0x2b, 0x02, 0xbe, 0x58, 0xbf, 0x0e, 0x1f, 0xb5,**

**0xbf, 0xba, 0x2e, 0xce**

*9.1* Sketch the organization of a three-way set associative cache with two-word blocks and a total size of 48 words. Your sketch should have a style similar to *Figure 5.18*, but clearly show the width of the tag and data fields.



*9.2* Trace the behavior of the cache from Exercise 9.1 Assume a true LRU replacement policy. For each reference, identify

• *the binary word address,*

• *the tag,*

• *the index,*

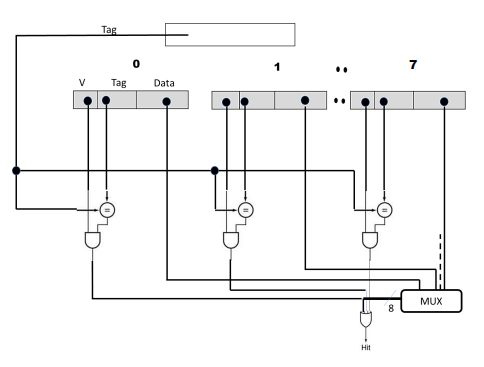
• *the offset*

• *whether the reference is a hit or a miss, and*

• *which tags are in each way of the cache after the reference has been handled.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hex Memory Reference | Binary Reference | Tag | Index | Offset | Hit/Miss |
| 0x03 | 0000 0011 | 0 | 1 | 1 | M |
| 0xb4 | 0000 0100 | b | 2 | 0 | M |
| 0x2b | 0010 1011 | 2 | 5 | 1 | M |
| 0x02 | 0000 0010 | 0 | 1 | 0 | H |
| 0xbe | 1011 1110 | b | 7 | 0 | M |
| 0x58 | 0101 1000 | 5 | 4 | 0 | M |
| 0xbf | 1011 1111 | b | 7 | 1 | H |
| 0x0e | 0000 1110 | 0 | 7 | 0 | M |
| 0x1f | 0001 1111 | 1 | 7 | 1 | M |
| 0xb5 | 1011 0101 | b | 2 | 1 | H |
| 0xbf | 1011 1111 | b | 7 | 1 | H |
| 0xba | 10111010 | b | 5 | 0 | M |
| 0x2e | 0010 1110 | 2 | 7 | 0 | M |
| 0xce | 1100 1110 | c | 7 | 0 | M |

*9.3* Sketch the organization of a fully associative cache with one-word blocks and a total size of eight words. Your sketch should have a style similar to Figure 5.18, but clearly show the width of the tag and data fields.



*9.4* Trace the behavior of the cache from Exercise 9.3. Assume a true LRU replacement policy. For each reference, identify

• *the binary word address,*

• *the tag,*

• *the index,*

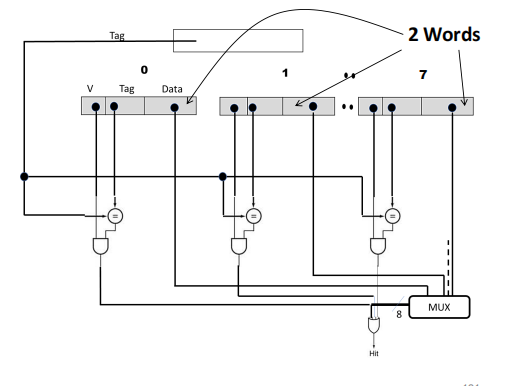
• *the offset,*

• *whether the reference is a hit or a miss*

• *the contents of the cache after each reference has been handled.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hex Memory Reference | Binary Reference | Tag | Hit/Miss | Content |
| 0x03 | 0000 0011 | 0x03 | M | 3 |
| 0xb4 | 0000 0100 | 0xb4 | M | 3, b4 |
| 0x2b | 0010 1011 | 0x2b | M | 3, b4, 2b |
| 0x02 | 0000 0010 | 0x02 | M | 3, b4, 2b,2 |
| 0xbe | 1011 1110 | 0xbe | M | 3, b4, 2b, 2, be |
| 0x58 | 0101 1000 | 0x58 | M | 3, b4, 2b, 2, be, 58 |
| 0xbf | 1011 1111 | 0xbf | M | 3, b4, 2b, 2, be, 58, bf |
| 0x0e | 0000 1110 | 0x0e | M | 3, b4, 2b, 2, be, 58, bf, e |
| 0x1f | 0001 1111 | 0x1f | M | b4, 2b, 2, be, 58, bf, e, 1f |
| 0xb5 | 1011 0101 | 0xb5 | M | 2b, 2, be,58, bf, e, 1f, b5 |
| 0xbf | 1011 1111 | 0xbf | H | 2b, 2, be,58, e, 1f, b5, bf |
| 0xba | 10111010 | 0xba | M | 2, be,58, bf, e, 1f, bf, ba |
| 0x2e | 0010 1110 | 0x2e | M | be,58, bf, e, 1f, bf, ba, 2e |
| 0xce | 1100 1110 | 0xce | M | 58, bf, e ,1f, bf, ba, 2e, ce |

*9.5* Sketch the organization of a fully associative cache with two-word blocks and a total size of eight words. Your sketch should have a style similar to Figure 5.18, but clearly show the width of the tag and data fields.



Similar to 9.3 except the data field is twice as large holding 2 words.

*9.6* Trace the behavior of the cache from Exercise 9.5. Assume an LRU replacement policy. For each reference, identify

• *the binary word address,*

• *the tag,*

• *the index,*

• *the offset,*

• *whether the reference is a hit or a miss,*

• *the contents of the cache after each reference has been handled.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hex Memory Reference | Binary Reference | Tag | Offset | Hit/Miss | Content |
| 0x03 | 0000 0011 | 0x01 | 1 | M | [2,3] |
| 0xb4 | 0000 0100 | 0x5a | 0 | M | [2,3], [b4,b5] |
| 0x2b | 0010 1011 | 0x15 | 1 | M | [2,3], [b4,b5], [2a,2b] |
| 0x02 | 0000 0010 | 0x01 | 0 | H | [b4,b5], [2a,2b],  [2,3] |
| 0xbe | 1011 1110 | 0x5f | 0 | M | [b4,b5],  [2a,2b],  [2,3],  [be,bf] |
| 0x58 | 0101 1000 | 0x2c | 0 | M | [2a,2b],  [2,3],  [be,bf],  [58,59] |
| 0xbf | 1011 1111 | 0x5f | 1 | H | [2a,2b],  [2,3],  [58,59],  [be,bf] |
| 0x0e | 0000 1110 | 0x07 | 0 | M | [2,3],  [58,59]  [be,bf],  [e,f] |
| 0x1f | 0001 1111 | 0x0f | 1 | M | [58,59]  [be,bf],  [e,f],  [1e,1f] |
| 0xb5 | 1011 0101 | 0xb5 | 1 | M | [be,bf],  [e,f],  [1e,1f],  [b4,b5] |
| 0xbf | 1011 1111 | 0xbf | 1 | H | [e,f],  [1e,1f],  [b4,b5],  [be,bf] |
| 0xba | 10111010 | 0xba | 0 | M | [1e,1f],  [b4,b5],  [be,bf],  [ba,bb] |
| 0x2e | 0010 1110 | 0x2e | 0 | M | [b4,b5],  [be,bf],  [ba,bb],  [2e,2f] |
| 0xce | 1100 1110 | 0xce | 0 | M | [be,bf],  [ba,bb],  [2e,2f],  [ce,cf] |

*9.7* Repeat Exercise 9.6 using MRU (most recently used) replacement.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hex Memory Reference | Binary Reference | Tag | Offset | Hit/Miss | Content |
| 0x03 | 0000 0011 | 0x01 | 1 | M | [2,3] |
| 0xb4 | 0000 0100 | 0x5a | 0 | M | [2,3], [b4,b5] |
| 0x2b | 0010 1011 | 0x15 | 1 | M | [2,3], [b4,b5], [2a,2b] |
| 0x02 | 0000 0010 | 0x01 | 0 | H | [b4,b5], [2a,2b],  [2,3] |
| 0xbe | 1011 1110 | 0x5f | 0 | M | [b4,b5],  [2a,2b],  [2,3],  [be,bf] |
| 0x58 | 0101 1000 | 0x2c | 0 | M | [b4,b5],  [2a,2b],  [2,3],  [58,59] |
| 0xbf | 1011 1111 | 0x5f | 1 | H | [b4,b5],  [2a,2b],  [2,3],  [be,bf] |
| 0x0e | 0000 1110 | 0x07 | 0 | M | [b4,b5],  [2a,2b],  [2,3],  [e,f] |
| 0x1f | 0001 1111 | 0x0f | 1 | M | [b4,b5],  [2a,2b],  [2,3],  [1e,1f] |
| 0xb5 | 1011 0101 | 0xb5 | 1 | M | [2a,2b],  [2,3],  [1e,1f],  [b4,b5] |
| 0xbf | 1011 1111 | 0xbf | 1 | H | [2a,2b],  [2,3],  [1e,1f],  [be,bf] |
| 0xba | 10111010 | 0xba | 0 | M | [2a,2b],  [2,3],  [1e,1f],  [ba,bb] |
| 0x2e | 0010 1110 | 0x2e | 0 | M | [2a,2b],  [2,3],  [1e,1f],  [2e,2f] |
| 0xce | 1100 1110 | 0xce | 0 | M | [2a,2b],  [2,3],  [1e,1f],  [ce,cf] |

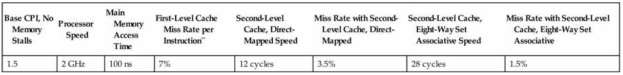
*9.8* Repeat Exercise 9.6 using the optimal replacement policy (i.e., the one that gives the lowest miss rate).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hex Memory Reference | Binary Reference | Tag | Offset | Hit/Miss | Content |
| 0x03 | 0000 0011 | 0x01 | 1 | M | [2,3] |
| 0xb4 | 0000 0100 | 0x5a | 0 | M | [2,3], [b4,b5] |
| 0x2b | 0010 1011 | 0x15 | 1 | M | [2,3], [b4,b5], [2a,2b] |
| 0x02 | 0000 0010 | 0x01 | 0 | H | [2,3], [b4,b5], [2a,2b] |
| 0xbe | 1011 1110 | 0x5f | 0 | M | [2,3], [b4,b5], [2a,2b] [be,bf] |
| 0x58 | 0101 1000 | 0x2c | 0 | M | [58,59], [b4,b5], [2a,2b], [be,bf] |
| 0xbf | 1011 1111 | 0x5f | 1 | H | [58,59], [b4,b5], [2a,2b], [be,bf] |
| 0x0e | 0000 1110 | 0x07 | 0 | M | [e,f], [b4,b5], [2a,2b], [be,bf] |
| 0x1f | 0001 1111 | 0x0f | 1 | M | [1e,1f], [b4,b5], [2a,2b], [be,bf] |
| 0xb5 | 1011 0101 | 0xb5 | 1 | H | [1e,1f], [b4,b5], [2a,2b], [be,bf] |
| 0xbf | 1011 1111 | 0xbf | 1 | H | [1e,1f], [b4,b5], [2a,2b], [be,bf] |
| 0xba | 10111010 | 0xba | 0 | M | [1e,1f], [ba,bb], [2a,2b], [be,bf] |
| 0x2e | 0010 1110 | 0x2e | 0 | M | [1e,1f], [ba,bb], [2e,2f], [be,bf] |
| 0xce | 1100 1110 | 0xce | 0 | M | [2a,2b],  [2,3],  [1e,1f],  [ce,cf] |

**10.** Multilevel caching is an important technique to overcome the limited amount of

space that a first-level cache can provide while still maintaining its speed. Consider a

processor with the following parameters:



*\*\*First Level Cache miss rate is per instruction. Assume the total number of L1 cache misses*

*(instruction and data combined) is equal to 7% of the number of instructions.*

*10.1* Calculate the CPI for the processor in the table using: 1) only a first-level cache, 2) a second-level direct mapped cache, and 3) a second-level eight-way set associative cache. How do these numbers change if main memory access time doubles? (Give each change as both an absolute CPI and a percent change.) Notice the extent to which an L2 cache can hide the effects of a slow memory.

Processor Speed: 2 GHz, which means the per cycle time is 0.5 ns.

Per access in main memory: 100/0.5 = 200 cycles.

1. 1.5 + 0.07\*200 = 15.5

2. 1.5 + 0.07 × (12 + 0.035 × 200) = 2.83

3. 1.5 + 0.07 × (28 + 0.015 × 200) = 3.67

Double the time:

1. 1.5 + 0.07\*400 = 29.5(90% increase)

2. 1.5 + .07 × (12 + 0.035 × 400) = 3.32(17% increase)

3. 1.5 + .07 × (28 + 0.015 × 400) = 3.88(5% increase)

*10.2* It is possible to have an even greater cache hierarchy than two levels? Given the

processor above with a second-level, direct-mapped cache, a designer wants to add a

third-level cache that takes 50 cycles to access and will have a 13% miss rate. Would this provide better performance? In general, what are the advantages and disadvantages of adding a third-level cache?

1.5 + 0.07 × (12 + 0.035 × (50 + 0.013 × 100)) = 2.47

Advantage: It will reduce the average access time by adding L3.

Disadvantage: It will increase the amounts of instructions and need more units.

*10.3* In older processors, such as the Intel Pentium or Alpha 21264, the second level of cache was external (located on a different chip) from the main processor and the firstlevel cache. While this allowed for large second-level caches, the latency to access the cache was much higher, and the bandwidth was typically lower because the second-level cache ran at a lower frequency. Assume a 512 KiB off-chip second level cache has a miss rate of 4%. If each additional 512 KiB of cache lowered miss rates by 0.7%, and the cache had a total access time of 50 cycles, how big would the cache have to be to match the performance of the second-level direct-mapped cache listed above?

Let x be the necessary miss rate.

1.5 + 0.07\*(50 + x\*200) < 2.83

Then, x <−0.155.

Since x is a negative number, no size will achieve the performance goal