IN1006 Systems Architecture 2024\_2025

Tutorial 05: Memory Hierarchy

Solutions

1. What is a memory hierarchy and why do we need one?

*The memory hierarchy is a way of organising memory in a computer to provide a large, cheap and fast storage system by placing small amount of fast expensive memory close to the processor, and cheap bulk but slower storage further away.*

*Why do we need a memory hierarchy? Processors have increased in speed faster than memory. Memory hierarchies can bridge the performance gap.*

1. Describe the two principles of locality that allow (data) memory caches to operate effectively.

* *Temporal locality: Items tend to get used more than once.*
* *Spatial locality: Items stored together get used together.*

1. What is the small fast memory that acts as a local buffer for the main memory called?

*Cache*

1. In the context of a cache, explain: cache hit and cache miss?

*When the processor requests data from memory it accesses the cache to see if it is already stored there. If it is then it is a cache hit. If it cannot retrieve the data then the processor loads the data it needs from main memory (DRAM) or a lower level-cache – a cache miss.*

1. Explain the operation of a direct mapped cache

*Key points are:*

* *Memory locations are mapped to unique cache positions.*
* *Cache Block = (Block Address) modulo (Number of Cache Blocks)*
* *Tag Bits: A tag at cache location gives high order bits*
* *Complete address = tag + cache address*
* *Valid Bit: Indicates whether the cache = memory*

1. Explain what happens during a cache miss

* *Stall the processor*
  + *Freeze contents of registers*
* *Activate memory controller*
  + *Separate controller from processor handles memory fetch*
* *Request data item from next lower level of hierarchy*
* *Load data item into cache*
  + *Write data, store upper bits as tag, set valid bit on.*
* *Resume the processor*

1. Describe the differences between combined and Harvard cache architectures

* *Split caches (a.k.a. Harvard Architecture)*
  + *Higher miss rate due to their smaller size*
  + *Higher bandwidth due to separate data paths*
  + *Data and instructions can be cached simultaneously*
* *Combined caches*
  + *Lower miss rate due to their size*
  + *Lower bandwidth due to sharing of data paths*
  + *Data and instructions cannot be cached simultaneously*

1. What is a Write Strategy? Give two examples

*A write strategy governs how and when a processor updates the cache and memory.*

* *Write through*
  + *Update cache and memory at the same time*
  + *Requires a buffer because the memory cannot accept data as fast as the processor can generate writes.*
  + *Processor must be stalled only if buffer is full*
* *Write back*
  + *Keep data in cache and write back when it is being replaced*
  + *Requires more sophisticated cache contents replacement unit, potentially increasing the cost of a cache miss*

1. Suppose a computer using direct mapped cache has 232 bytes of byte-addressable main memory, and a cache of 1024 blocks, where each cache block contains 32 bytes.
2. How many blocks of main memory are there?
3. What is the format of a memory address as seen by the cache, i.e., what are the sizes of the tag, block, and offset fields?
4. To which cache block will the memory address 0x000063FA map?

*a) 232/25 = 227*

*b) 32 bit addresses with 17 bits in the tag field, 10 in the block field, and 5 in the offset field*

*c) 000063FA = 00000000000000000 1100011111 11010, which implies Block 799*

**10** Suppose we have a computer that uses a memory address word size of 8 bits. This computer has a 16-byte cache with 4 bytes per block (🡪 4 blocks). The computer accesses a number of memory locations throughout the course of running a program. Suppose this computer uses direct-mapped cache. The format of a memory address as seen by the cache is given below:

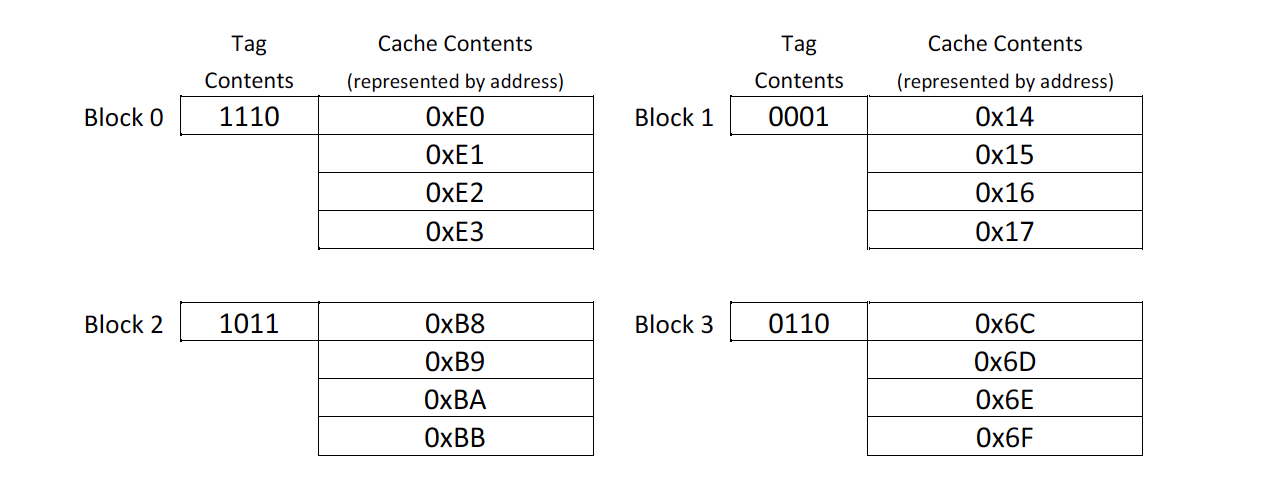
Tag: 4 bits

Block: 2 bits

Offset: 2 bits

The memory addresses of the first four accesses have been loaded into the cache blocks as shown below. (The contents of the tag are shown in binary and the cache “contents” are simply the address stored at that cache location.)

1. What is the hit ratio for the entire memory reference sequence given above, assuming we count the first four accesses as misses?
2. What memory blocks will be in the cache after the last address has been accessed?
3. What is the effective access time EAT if AccessC=10ns and AccessMM=200ns?

**

*Answer:*

*(a)*

To work out the answer, the hexadecimal memory addresses are converted to binary and then the contents of words in the relevant blocks in cache are checked. Also, after transforming hexadecimal addresses to binary, we have used the following colouring to distinguish the tag, block and offset within the binary address.

Tag: 4 bits

Block: 2 bits

Offset: 2 bits

The system accesses memory addresses in this exact order:

**0x6E** 🡪 0110 11 10 (11 🡪 block 3, 0110 🡪 tag, 10 🡪 2nd word): **miss**, leading to loading of block 3 as indicated above

**0xB9** 🡪 1011 10 01 (10 🡪 block 2, 1011 🡪 tag, 01 🡪 1st word): **miss**, leading to loading of block 2 as indicated above

**0x17** 🡪 0001 01 11 (01 🡪 block 1, 0001 🡪 tag, 11 🡪 3rd word): **miss**, leading to loading of block 1 as indicated above

**0xE0** 🡪 1110 00 00 (00 🡪 block 0, 1110 🡪 tag, 00 🡪 0th word): **miss**, leading to loading of block 0 as indicated above

**0x4E** 🡪 0100 11 10 (11 🡪 block 3, 0100 🡪 tag, 10 🡪 2nd word): **miss** as block 3 does not have 0100 tag

Due to the miss, the contents of block 3 will be replaced and become

|  |  |  |
| --- | --- | --- |
| Block 3 | Tag 0100 | 4C (0th word) |
| 4D (1st word) |
| 4E (2nd word) |
| 4F (3rd word) |

**0x4F** 🡪 0100 11 11 (11 🡪 block 3, 0100 🡪 tag, 11 🡪 3rd word): **hit, as it was loaded in previous step**

**0x50** 🡪 0101 00 00 (00 🡪 block 0, 0101 🡪 tag, 00 🡪 0th word): **miss** as block 0 does not have 0101 tag

Due to the miss, the contents of block 0 will be replaced and become

|  |  |  |
| --- | --- | --- |
| Block 0 | Tag 0101 | 50 (0th word) |
| 51 (1st word) |
| 52 (2nd word) |
| 53 (3rd word) |

**0x91** 🡪 1001 00 01 (00 🡪 block 0, 1001 🡪 tag, 01 🡪 1st word): **miss** as block 0 does not have 1001 tag

Due to the miss, the contents of block 0 will be replaced and become

|  |  |  |
| --- | --- | --- |
| Block 0 | Tag 1001 | 90 (0th word) |
| 91 (1st word) |
| 92 (2nd word) |
| 93 (3rd word) |

**0xA8** 🡪 1010 10 00 (10 🡪 block 2, 1010 🡪 tag, 00 🡪 0th word): **miss** as block 2 does not have 1010 tag

Due to the miss, the contents of block 0 will be replaced and become

|  |  |  |
| --- | --- | --- |
| Block 2 | Tag 1010 | A8 (0th word) |
| A9 (1st word) |
| AA (2nd word) |
| AB (3rd word) |

**0xA9** 🡪 1010 10 01 (10 🡪 block 2, 1010 🡪 tag, 01 🡪 1st word): **hit,** see contents of block 2 as modified in the previous access attempt

**0xAB** 🡪 1010 10 11 (10 🡪 block 2, 1010 🡪 tag, 11 🡪 3rd word): **hit,** see contents of block 2 as modified in the access attempt for A8 above

**0xAD** 🡪 1010 11 01 (11 🡪 block 3, 1010 🡪 tag, 01 🡪 1st word): **miss** as block 3 does not have 1010 tag

Due to the miss, the contents of block 3 will be replaced and become

|  |  |  |
| --- | --- | --- |
| Block 3 | Tag 1010 | AC (0th word) |
| AD (1st word) |
| AE (2nd word) |
| AF (3rd word) |

**0x93** 🡪 1001 00 11 (00 🡪 block 0, 1001 🡪 tag, 11 🡪 3rd word): **hit**, block 0 has this memory address after its last modification

**0x94** 🡪 1001 01 00 (01 🡪 block 1, 1001 🡪 tag, 10 🡪 0th word): **miss** as block 1 does not have 1001 tag

Due to the miss, the contents of block 1 will be replaced and become

|  |  |  |
| --- | --- | --- |
| Block 1 | Tag 1001 | 94 (0th word) |
| 95 (1st word) |
| 96 (2nd word) |
| 97 (3rd word) |

(b)

Block 0 will be as it became after the attempt to access the memory address **0x91** above.

Block 1 will be as it became after the attempt to access the memory address **0x94** above.

Block 2 will be as it became after the attempt to access the memory address **0xA8** above.

Block 3 will be as it became after the attempt to access the memory address **0xAD** above.

(c) Altogether there were 14 cache memory access attempts. Of these 10 were misses and 4 were successful (i.e., hits). Hence, H = 4/14 = 0.286 and M = 1 – H = 0.714. Thus,

EAT = H × AccessC + (1-H) × AccessMM =

= 0.286 \* 10ns + (1-0.286) \* 200ns =

= 2.86ns + 142.8ns = 145.66ns