**(Exercise) Direct-map cache**

**TPS (Think-Pair-Share) activity 1 Paired with the classmate sitting next to you and answer the following questions (30 minutes):**

1. What is cache? Why do we need cache?

Cache is a small on-chip memory that has copies of likely to be accessed memory items for the fast access of these data items.

2. There are generally 2 practical ways to organize a cache: Direct-mapped cache and N-way set associative cache. In both types of cache, data at a specific address of the main memory (RAM) are mapped to a pre-defined location in the cache. A “Block” is the basic unit of data being mapped between the main memory and cache. The size of a block depends on the specification of a cache. Every time data are transferred between cache and the main memory, it is a block of data being transferred. In this exercise, we will explore the Direct-mapped cache.

3. In a Direct-mapped cache, the cache is organized as a hash table. Addresses of the main memory are mapped to the indices of the cache (block numbers) using a modulo operator (%) as the hash function. As a result, we can divide a memory address into 3 fields: tag, index, offset.

4. Offset bits tell us how many bytes of data are in a block. These bits are the right-most bits of the memory address. You can consider this as the number of columns of data in a cache. With a specific value of the offset bits from an address, we know which column of a block we are trying to access. Given the block size of a cache is 16B (bytes), how many bits do we need for offset? What is the number of bits in offset as a function of block size? Is it practical to have a cache of block size = 1 byte?

For a 16 Byte Cache we know that to obtain the offset we need to do the following:

2(Block) = bits for the offset OR 2n = bits for the offset

Since our block is 16 Bytes, then we calculate with this function it:

2(16) = 4 bits for the offset

Having a cache of block size 1 byte :

2(1) = 0 bits for the offset

As we can observe, having a 1 byte block would be the most impractical implementation of cache; you basically can only story one data item in there and this behaviour is not useful.

5. Index bits tell us how many blocks there are in a cache. These bits are the next right-most bits of the memory address after offset bits. You can consider this as the number of blocks (rows) of data in a cache. With a specific value of the index bits from an address, we know which block (row) we are trying to access. Given there are 64 blocks in a cache, how many index bits do we need? What is the number of bits in index as a function of number of blocks?

We can express the function to obtain our index bits as follows:

Number of blocks Associativity = Index bits

Right off the bat, we know that the associativity of a direct mapped cache is 1, given this info we can compute the following:

64 blocks 1 Associativity = 64 = 26 = 6 bits

Which can also be written as:

2(number of blocks OR rows) = bits

6. Once you know the number of blocks and the block size of a cache, do you know the total size of the cache? How?

Yes we do, we have 64 blocks and 16 bytes so we multiply the number of bytes from

The blocks times the number of bytes of the block:

2624 = 26+4 = 210

10 bits

7. Since the size of cache is always smaller than the size of the main memory, the sum of bits of the offset and index of a cache will be less than the number of bits in an address of the main memory. What do we do to the left over bits from the address? Why are they important?

We utilize the remaining bits for the tag:

2n total-210

Which is utilized to check if it is a hit or miss

8. Given a memory address of 20 bits (during Intel 8086 era), 128B of cache, and 8B block size, answer the following questions:

a. How big is this main memory?

To find this we utilize the memory address and find it 220

b. How many offset bits?

23 = 8B block so offset is 3 bits

c. How many blocks are there in the cache?

128B cache8B block = 16 blocks/rows

d. How many index bits?

24 = 16 blocks = 4 bits

e. How many tag bits?

Address - Offset + Index = Tag → 20 - 3 + 4 = 13 bits for tag

f. Draw the layout of the cache: including tags, valid bits, dirty bits, and data blocks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Valid Bit | Dirty Bit | Index bits | Offset Bits | Tag bit |
| 0 | 0 | 0000 | 000 | 0 0000 0000 0000 |
| 0 | 0 | 0001 | 001 | 0 0000 0000 0001 |
| 0 | 0 | 0010 | 010 | 0 0000 0000 0010 |
| 0 | 0 | 0011 | 011 | 0 0000 0000 0011 |
| 0 | 0 | 0100 | 100 | 0 0000 0000 0100 |
| 0 | 0 | 0101 | 101 | 0 0000 0000 0101 |
| 0 | 0 | 0110 | 110 | 0 0000 0000 0110 |
| 0 | 0 | 0111 | 111 | 0 0000 0000 0111 |
| 0 | 0 | 1000 | 000 | 0 0000 0000 1000 |
| 0 | 0 | 1001 | 001 | 0 0000 0000 1001 |
| 0 | 0 | 1010 | 010 | 0 0000 0000 1010 |
| 0 | 0 | 1011 | 011 | 0 0000 0000 1011 |
| 0 | 0 | 1100 | 100 | 0 0000 0000 1100 |
| 0 | 0 | 1101 | 101 | 0 0000 0000 1101 |
| 0 | 0 | 1110 | 110 | 0 0000 0000 1110 |
| 0 | 0 | 1111 | 111 | 0 0000 0000 1111 |

g. What is the number of bits per row of the cache (number of bits being used in a row: tag, valid bit, dirty bits, and data block)?

Bits per row is:

Valid bit + Dirty bit + Tag bits + cache Bytes x Byte size is always 8 bytes

1 bits + 1 bits + 13 bits + 8 Bytes x 8 Byte

15 bits + 8 Bytes x 8 Byte

2 bits + 3 Bytes + 8 Bytes x 8 Bytes

15 bits + 64 bits

79 bits per row

**(Exercise) N-way set associative cache**

**TPS (Think-Pair-Share) activity 2 Paired with the classmate sitting next to you**

**and answer the following questions (30 minutes):**

1. What is the disadvantage of a Direct-mapped cache? What kind of cache miss will it introduce?

The disadvantage with direct mapped cache is that each time a memory block doesn’t exist in the cache the system copies the block into the address, replacing whatever was there previously. Sometimes it will continually will replace itself many times causing **thrashing** in the memory. This kind of miss is known as **thrashing.**

2. To overcome this problem, we can allow multiple blocks of data to occupy the same set of a cache. Note that we use “set” here instead of index of cache. In this organization, we group N blocks (rows) of cache into a set and allow more than one block of data to stay within a set. The layout of the cache remains the same as its direct-mapped version, but the difference is that every N blocks are now being grouped into a set.

3. The memory address is still partitioned into the same 3 fields, but the index bits now refer to the set number. Given a cache with 1024 blocks and the associativity is 4 (4 blocks per set), how many index bits do we need? What is the number of bits in index as a function of number of blocks and associativity?

Number of blocks Associativity = 2n  = n Index bits OR

2(Number of blocks Associativity) = bits

1024 blocks 4-way set associative = 256(each block) = 28 = 8 bits

4. Given a memory address of 20 bits (during Intel 8086 era), 128B of 2-way cache, and 8B block size, answer the following questions:

a. How big is this main memory?

The main memory is 220 bytes

b. How many offset bits?

Offset bits is **3** since 23(the bit) = 8 bytes

c. How many blocks are there in the cache?

128B 8 B = 16 blocks of 2-way associative

d. How many sets are there in the cache?

128 Bytes 8 Bytes 2-way associative = 8 sets (OR blocksassociativity)

e. How many index bits?

23 = 8 sets = 3 bits

f. How many tag bits?

Address - Offset + Index = Tag → 20 - 3 + 3 = 14 bits for tag

g. Draw the layout of the cache: including tags, valid bits, dirty bits, and data blocks. Indicate the sets with a different color (or a thicker) border.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Valid Bit | Dirty Bit | Index bits | Offset Bits | Tag bit |
| 0 | 0 | 000 | 000 | 0 0000 0000 0000 |
| 0 | 0 | 001 | 001 | 0 0000 0000 0001 |
| 0 | 0 | 010 | 010 | 0 0000 0000 0010 |
| 0 | 0 | 011 | 011 | 0 0000 0000 0011 |
| 0 | 0 | 100 | 100 | 0 0000 0000 0100 |
| 0 | 0 | 101 | 101 | 0 0000 0000 0101 |
| 0 | 0 | 110 | 110 | 0 0000 0000 0110 |
| 0 | 0 | 111 | 111 | 0 0000 0000 0111 |
| 0 | 0 | 000 | 000 | 0 0000 0000 1000 |
| 0 | 0 | 001 | 001 | 0 0000 0000 1001 |
| 0 | 0 | 010 | 010 | 0 0000 0000 1010 |
| 0 | 0 | 011 | 011 | 0 0000 0000 1011 |
| 0 | 0 | 100 | 100 | 0 0000 0000 1100 |
| 0 | 0 | 101 | 101 | 0 0000 0000 1101 |
| 0 | 0 | 110 | 110 | 0 0000 0000 1110 |
| 0 | 0 | 111 | 111 | 0 0000 0000 1111 |

h. What is the number of bits per row of the cache (number of bits being used in a row: tag, valid bit, dirty bits, and data block)?

Valid bit + Dirty bit + Tag bits + cache Bytes x Byte size is always 8 byte

1 +1 + 14 + 8 x 8

16 + 64

80 bits

**(Assignment, individual) Cache in your computer**

**Download and install CPUID and find out details about the cache(s) in your computer.**

**For Windows: CPU-Z: https://www.cpuid.com/softwares/cpu-z.html**

**For Mac: MacCPUID: https://software.intel.com/en-us/download/download-maccpuid**

**For Linux: https://www.tecmint.com/check-linux-cpu-information/**

**Answer the following questions:**

1. How many levels of caches does your CPU have (L1, L2, L3, etc.)? Is there separate L1 cache for data and instructions?

My pc has 3 levels of cache(L1, L2, L3)

and it does have L1d cache for data and L1i for instructions.

2. How big is each level of cache?

L1d cache: 32K

L1i cache: 32K

L2 cache: 256K

L3 cache: 3072K

3. What is the block size (sometimes it is called line size)?

My Machine has a block size of 64 Bytes

4. Are the caches direct-mapped or set associative? If set associative, how many ways?

My Machine is set associative:

L1d cache: 8-way Set-associative

L1i cache: 8-way Set-associative

L2 cache: 4-way Set-associative

L3 cache: 12-way Set-associative

5. With L1 data cache, how many tag bits, index bits, and offset bits?

Offset bits : 2( 64 ) = 6 bits

Index bits : 2([3072000 Bytes 64Bytes]4-way associative) = 4 bits

Tag bits : 64bits - 6 bits + 4 = 54bits