COMPSCI 2GA3 Assignment 4

Elite Lu lue13 Task 1

1. The requirements for a component to qualify as a cache is that it is small, active, transparent, and automatic. To start off, the size of the cache must be smaller than the size of the data at the producer. For active, the component must contain an algorithm that does both decision making and request handling. Transparency means that the producer and consumer are unaware of its existence. The component must be automatic meaning that it is not controlled by an external mechanism.

We measure cache performance by using the cost of a hit and a cost of a miss along with the hit ratio. The formula is written as follows:

Ccache= r x Ch + (1 – r) x Cm

In order to calculate for the value of r, which is the ratio, the value is shown below:

r = Nhit/(Nhit + Nmiss)

The expected benefit of using a cache is that the cost of lookup is a lot faster. Adding multiple caches is essential for quicker lookup times.

1. In order to calculate the number of bits to represent the tag of each of these blocks, I will first determine the number of bits and using the number of words per line and the number of blocks. Because there are 32 bits per line and it is implemented in a 16-bit architecture (2 bytes), that means there are 16 words. In addition to this, there are 8 blocks. The fact that both are powers of two is important since it means that I can do division via bit shifting when determining the actual values. 16 is equivalent to 24 and 8 is equivalent to 23. With this, I will subtract 4 and 3 from 16 bits, which will get me 9 bits. Therefore, there will be 9 bits for the tag ID.

Here is how the cache translate the information. It will first get a word from a block. An example would be 0x50, which is 80 in decimal. This can also be represented in binary as 01010000. I will first determine the offset by using the formula o = a mod W, where a is the address and W is the words per line. This would equal 0, as 80 mod 8 is 0. Next, I will find the block ID. The block ID uses a similar algorithm, where b = (a div W) mod B, where B is the number of blocks/lines in the cache. There are 8 blocks of memory, so with the equation being b = (80 div 8) mod 8, I can determine the value. Because B and W are both powers of 2, I can bit shift to get 2. 2 is represented as 010, since I need 3 bits to represent all 8 blocks. Finally, I determine the tag ID by using the equation t = (a div W) div B, which results in the tag ID being 1, or rather 000000001, as in the previous part I determined the tag ID is 9 bits long. Finally, the translation is shown below:

000000001 010 0000

1. To calculate the physical memory size of each page in the system, I will first be required to determine how physical memory is comprised. Physical memory is made up of frame storage, page tables, and the operating system. To calculate this, I will simplify each of these values into exponents of 2:

32-bit = 25-bit

256 MB = 220 x 28 = 228 bytes

128 MB = 220 x 27 = 227 bytes

1024 entries = 210 entries (pages)

1024 frames = 210 frames

Next, in order to calculate the amount of physical storage per page, I will first subtract the total storage of RAM with the amount of storage blocked, which results in there only being 128 MB for the set of 1024 entries on the page table. After that, I will divide the amount of storage set for the table by the number of pages. This will result in 17 bytes per page, which is 217 bytes for storage.

The MMU performs this address translation by talking to each physical memory in the physical address space. It translates multiple virtual spaces to multiple physical spaces. It does this by first getting the virtual address. Then, it will calculate the page number with the formula N = a div P, where P is the number of bytes. Then, the frame address is fetched from the page table by using the page number as the index. To calculate the offset, I will perform a similar function O = a mod P. Finally, the physical address is translated by combining the frame address and the offset.

A Translation Lookaside Buffer, or a TLB, is a special high-speed hardware device that is used for optimizing the performance of a paging system. It can help speed up translation since it optimizes successive lookups by avoiding indexing into a page table.

# Bonus

For the bonus, I will explain this output for the node shown below:

0x0000016b2c3378 a0 0f 00 00 01 00 00 00 88 33 2c 6b 01 00 00 00

For this, the first part, 0x0000016b2c3378, is the pointer to the variable. The next 4 bytes represent the actual value. The bytes are a0 0f 00 00. However, because this is little endian, I must reverse it to get the actual value, which is 4000. The next four bytes are useless, as those are meant for alignment. Finally, I have the last 8 bytes, which represent the pointer to the next variable. Like the previous part, the pointer is stored in little endian.

Similarly, the print function can be used for the integer char array and the float ones. This is the one for the integer array:

0x00000159e06790 f0 d8 ff ff

The first part is the pointer to the variable and the next four bytes are the byte values for the integer. The same can be said for the float array. Both are shown in the test cases.