CSCI-UA.0201

Computer System Organization

Homework Assignment 3

1) Consider a computer system that has a cache with 4096 blocks. Each block can store 16 bytes. What will be the value stored in the *TAG* field of the cache block that holds the memory block containing the address 0xABCDEF (This hexadecimal number gives you a hint on the address length):

(i) [2 points] **if it is a direct-mapped cache**

block size = 16 = 2^4 so offset is 4 bits -> 1 hexadecimal digit
#sets = #blocks/associativity = 4096/1 = 2^{12} so sets field is 12 bits -> 3 hex digits
Tag is then the leftmost two digits from the addres = 0xAB

(ii) [2 points] **if it is a 16-way set-asscociative cache**

block size is same as above
#sets = 4096/16 = 2^8 so sets field is 8 bits -> 2 hex digits
Tag is then the leftmost 3 digits -> 0xABC

(iii) [1 point] **if it is fully associative**

block size same as above
No sets field
TAG is then all the address except the rightmost hex digit ->0xABCDE

2. Consider a small 2-way set associative cache with a total of 32 blocks and a block size of 256 bytes. The cache uses LRU replacement policy. Assume that the cache is initially empty. The CPU accesses the following memory locations, in that order: 0x55c88, 0x55774, 0x5479c, 0x54c00, 0x55784, 0x56c80, 0x56718, 0x54738.

(a) [3 points] **How the address is split to do cache lookup?**

Tag	Set	Offset

Each address is 5 hexadecimal digits. Each hexadecimal digit is represented with 5 binary bits. Hence, the total address length is 20 bits.
Block size = 256 = 2^8 → Offset is 8 bits
#sets = Cache size / set size = 32/2 = 16 → Set index is 4 bits
Therefore, the tag is 8 bits.

It is easy now to see that the most significant two hexadecimal digits represent the tag, the following digit is the set, and the last two digits are the offset. For example, for the address 0x55c88, the tag is 55, the set index is c, and the offset is 88. This will simplify the following part of the problem

(b) [8 points] **For each memory reference, indicate whether it will result in hit or miss and, if miss, indicate the type of miss (compulsory, capacity, or conflict)**

Address	**TAG**	**Set**	**Offset**	**H/M**	Comments
0x55c88	55	c	88	M	Compulsory miss
0x55774	55	7	74	M	Compulsory miss
0x5479c	54	7	9c	M	Compulsory miss
0x54c00	54	c	00	M	Compulsory miss
0x55784	55	7	84	H	
0x56c80	56	c	80	M	Replaces block with Tag 55
0x56718	56	7	18	M	Replaces block with Tag 54
0x54738	54	7	38	M	Replaces block with Tag 55

(c) [1 point] **From your solution in part (b) above, what is the hit rate of the cache?**

The memory is accessed 8 times and had only 1 hit, as indicated by the table above. This gives a hit rate of 1/8 = 0.125

(d) [3 points] **Assume the memory access latency is 10 cycles and the cache, mentioned in this problem, has an access latency of 3 cycles. Also assume the hit rate**

you calculated in part (c) above. Did this system benefit from having a cache? Justify

(partial credits for this problem are noted next to the arrows)

If we did not have cache, then each access will take 10 cycles. ← 0.5 point

With a cache, the average access time = m + (1-p)M = 3 + (1-0.125)10 = 11.75 ← 1.5 point

This is larger than accessing the memory directly. So the system did not benefit from the cache. ← 1 point

3. **Almost all programs need external libraries.**
   (a) In how many places can needed libraries be added to your code?

   In 3 places

   (b) What are these places?

   - statically during linking
   - dynamically during loading
   - dynamically during execution

   (c) For each one of those places, state one advantage and one disadvantage.
   - statically during linking
     - +: The executable is self-content. So, if you run it on a different machine, you do not have to worry about the version of the installed libraries or whether they are installed.
     - - : The executable file becomes much larger in size.
     - -: Is not very efficient in terms of memory usage.
   - dynamically during loading
     - +: The executable file is smaller.
     - +: Makes better use of the memory because libraries are shared among programs.
     - -: Loading becomes slower.
   - dynamically during execution
     - + Better memory usage because libraries are loaded only when needed.
     - + loading time is not slower.
     - + The executable file is not large.
     - - Puts a burden on the programmer.
     - - Slowdown during execution to load needed libraries

4.
```
int array1[M][N];
int array2[N][M];

int copy(int i, int j)
{
    array1[i][j] = array2[j][i];
}
```

Suppose the above code generates the following assembly code (assume array2 and array1 are the base addresses of the corresponding arrays):

```
copy:
        movl %rdi, %ecx                     → ecx = i
        movl %rsi, %ebx                     → ebx = j
        leal (%ecx,%ecx,8), %edx            → edx = 9*ecx = 9i
        sall $2, %edx                       → edx = edx*4 = 36i
        movl %ebx, %eax                     → eax = ebx = j
        sall $4, %eax                       → eax = eax*16 = 16j
        subl %ebx, %eax                     → eax = eax – ebx = 16j – j = 15j
        sall $2, %eax                       → eax = eax *4 = 60j
        movl array2(%eax,%ecx,4), %eax      → eax = arra2+60j+4i
        movl %eax, array1(%edx,%ebx,4)      → array1+36i +4j = eax
        ret
```

What are the values of M and N: **M= 15 and N = 9**

Show how did you reach your answer:

- An int is 4 bytes
- See the explanation next to the assembly code above
- Accessing array1[i][j] = array1 + (#elements in a row * 4*i) + (j*4) = array1+4Ni+4j →array2+36i + 4j → N = 9
- Same array2[j][i] = array2 +4Mj+4j → array2+60j + 4i → M = 15

5. From the assembly code and since both fun1 and fun2 have the same arguments in the same order:

- edx → a
- eax → b
- By convention, the result of the function is returned in eax
- cmpl %eax, %edx → edx – eax → a –b
- Combining the above bullet with jg2 .L9 we find that it does:
  if(a >= b) return b → if (a < b) return b → **so fun1**

6.

Hexadecimal	**binary**	**Decimal (assuming unsigned)**	**Decimal (assuming signed)**
0x8A	1000 1010	2+8+128 = 138	-(2s compl of num) = -(01110110) = -(64+32+16+4+2) = - 118
0x21	0010 0001	32+1 = 33	33