|  |  |
| --- | --- |
| **CSID #1** | **Y7m1b** |
| **CSID #2** |  |

**Part 1: 1a**

Place the letters CO, CI, or CT into each box indicating if the corresponding bit number is part of the cache block offset (CO), the cache set index (CI), or the cache tag (CT).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CT | CT | CT | CT | CT | CT | CT | CT | CT | CI | CI | CI | CO | CO | CO |

**Part 1: 1b**

For each row, fill in the cache set index, cache tag, and value with a hexadecimal number. In the Hit column, place an H if the address produces a hit and M if the address produces a miss.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Address | Cache Set Index | Tag | Hit | Value |
| Read | 0x0117 | 2 | 0x4 | M | 94 |
| Read | 0x64C2 | 0 | 0x193 | H | 99 |
| Read | 0x7BFE | 7 | 0x1EF | M |  |
| Read | 0x0110 | 2 | 0x4 | M | 5F |

**Part 3:**

For each scenario, enter the miss rate as a percentage and then explain why the miss rate is what it is.

|  |  |  |
| --- | --- | --- |
| Scenario | Miss Rate | Explanation |
| A | 25% | 16 bytes each block, when an int is 4 bytes. This means that a block will house 4 continuous integers. In Scenario A we access elements in row major order which follows the way arrays are stored.  As a result, every time we pull a block due to a miss we will obtain 3 hits because of it translating to a miss rate of ¼ = 25%. |
| B | 100% | We are accessing in column major order. 4 Rightmost bits are the offset, the next 7 bits indicate index. A given element in row x has address y, the element in row x + 1 has address y + 256. This means there are only 8 different indexes that data will be put into, so on the 9th iteration the first block retrieved will be overwritten by a new block. As a result, each access of the array will be a miss leading to a 100% miss rate |
| C | 50% | The loop structure accesses the array in column order, however because we use the next element in terms of row order each time, we get to use the cache more than scenario B.  Array is accessed a(I,j) a (i+1, j) a(i,j+1) a(i+1,j+1)  First two accesses the array in column order, the last two accesses however are in row order which will be hits from the previous 2 accesses. Which gives a hit rate of 2/4 = 50%. |
| D | 25% | This gives 25% aka the best possible miss rate for all different sum functions. This implies that the change of size from 64X64 to 68X68 makes the order of retrieval inconsequential. In scenario B only 8 out of 128 lines are used. But here 68 lines, 1 for each column. So we will miss 68 times first then we will hit 68\*3 = 204 times leading to a miss rate of 68/272 = 25%. |

**Part 4:**

Enter your log entries in this table.

|  |  |  |  |
| --- | --- | --- | --- |
| Log # | Average elapsed time | Time relative to baseline | Explanation |
| 1 | 1982673 | 1 | Fill in your baseline results in this row. |
| 2 | 1790661 | 0.931 | Combined first and second loops, takes advantage of potentially cached values in the same row. |
| 3 | 1588067 | 0.802 | Immediate left combined with above and top left. Continuation of the same rationale as above. |
| 4 | 1447433 | 0.700 | Combined top right loop with immediate right. Same Rationale as above. |
| 5 | 1193761 | 0.601 | Combined bottom left with immediately below loop. Same Rationale as above. |
| 6 | 999578 | 0.503 | Combined bottom left with bottom right. Had to introduce a new variable col1 because the different columns are accessed in different orders |
| 7 | 636646 | 0.320 | Combined all calc loops into one created two different column variables as well as two different row variables. |
| 8 | 363872 | 0.183 | Combined ending and initializing loops. Seems like the end of combining loop optimization. |
| 9 | 352865 | 0.178 | used local variables instead of writing directly to new world double array each time. |
| 10 | 345085 | 0.173 | Reduce writing even more using local variables |
| 11 | 62595 | 0.0293 | Swapped the inner and outer loops. So access arrays in a row major form. |
| 12 |  |  |  |
| 13 |  |  |  |
| 14 |  |  |  |
| 15 |  |  |  |
| 16 |  |  |  |
| 17 |  |  |  |
| 18 |  |  |  |
| 19 |  |  |  |
| 20 |  |  |  |
| 21 |  |  |  |