Ian Zhang

Professor Alvin Lebeck

Computer Science 250

Assignment #5

1. (15 pts) Cache Organization

Consider a CPU that has a maximum of 4GB addressable main memory. This CPU

has a Level 1 cache of 64K Bytes with 128 byte block size. For each of the following

associativities:

a. direct mapped

b. 4way set associative

c. fully associative

Show the size of the tag, index and offset fields in a memory address, and compute

the total number of bits used to implement each of the three cache configurations.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Cache Type** | **Memory Address Bits** | **Cache Size** | **Block/Line Size (Bytes)** | **Cache Sets** | **Tag** | **Index** | **Offset** | **Total Bits** |
| Direct Mapped | 4GB main mem = 32 | 64000 | 128 | 500 | 16 | 9 | 7 | 532,992 |
| 4-Way Set Associative | 4GB main mem = 32 | 64000 | 128 | 125 | 18 | 7 | 7 | 534,016 |
| Fully Associative | 4GB main mem = 32 | 64000 | 128 | 1 | 25 | 0 | 7 | 537,600 |

Cache Type, Mem Address, Cache Size, Block Size Given.

Cache sets = Cache Size / # lines

Log(line size) = offset bits

Log(cache sets) = bits for cache index

Remaining upper bits = tag address bits

Total bit calculation:

**Direct Mapped**

Bits Per Block \* # of Blocks

Bits Per Block: data bits + tag bits + valid bit

Data bits = 128 bytes \* 8 bits = 1024 bits per block

Tag bits = 16

Valid bit = 1

500 \* (1024 + 16 + 1) = 532,992 bits

**4-Way Set Associative**

Data bits = 128 bytes \* 8 bits = 1024 bits per block

Tag bits = 18

Valid bit = 1

500 \* (1024 + 18 + 1) = 534,016 bits

**Fully Associative**

Data bits = 128 bytes \* 8 bits = 1024 bits per block

Tag bits = 25

Valid bit = 1

500 \* (1024 + 25 + 1) = 537,600 bits

2. (5 pts) Describe the general characteristics of a program that exhibits very little temporal and very little spatial locality with regard to data accesses. Provide an example program (pseudocode is acceptable).

/\* This program reads 1000 files and writes them to output files \*/

//select first file

//jump random 3GB in memory address away to function that processes open file

//open file

//jump 3GB to function that processes read file

//read file

//jump random 2GB in memory address away to function that writes data

//write data

//…

//end

This program exhibits very little temporal and very little spatial locality with regard to data accesses. This exhibits low spatial locality because each time data is accessed (on open/read/write), the code does not consider that consecutive addresses in memory may need to be used, jumping all over the place in storage to find the needed next-step to process. This extends access time significantly. This exhibits low temporal locality as well because each time data is accessed, the code does not consider that the code just executed (i.e., read file) could be stored in cache and used 999 more times without any additional access time required. Instead, it moves from one block of code to another without consideration of its potential reuse in the next line of code.

3. (5 pts) Describe the general characteristics of a program that exhibits high temporal

locality but very little spatial locality with regard to data accesses. Provide an

example program (pseudocode is acceptable).

/\*

\*This program is a terribly implemented linked list

\*that traverses a short list of elements that are very far

\*away in memory several times.

\*Loops 6 times for 3 nodes in linked list

\*/

//create struct

//establish root node

//find memory address of next node

//jump random 2 GB to that memory address

//point to this node from root

//reestablish pointers

//find memory address of next node

//jump random 3 GB to that memory address

//point to this node from previous node

//loop through list

//end

This program exhibits very little spatial locality but high temporal locality with regard to data access. There is very little spatial locality exhibited in this program because the locations in memory of the two nodes are extremely far apart and randomly addressed in memory. So each time the machine must expensively look far and wide for these locations. However, there is high temporal locality exhibited in this program because of the loop constructor—which assumes code just used to traverse the list has a high likelihood of being used again—thus the loop runs through the same traversal multiple times, accessing the same code each time, relying on the fact that it will be looped through again soon as a method of decreasing access time.

4. (15 pts) Average Memory Access Time

Find the average memory access time (t\_avg) for a machine with:

a. A single level cache with a one cycle hit time, a 300 cycle miss penalty, a 5% miss rate, and a 2ns clock.

tavg = thit + %miss \* miss penalty

tavg = 2 + .05 \* 300 = 17 ns

b. A two level cache hierarchy with a two cycle L1 hit time, a 12 cycle L2 hit time, a 100 cycle L2 miss penalty, a 1ns clock, a 10% L1 miss rate, and a 1% L2 miss rate.

tavg = 2 + .1 \* (12 + .01 \* 100) = 3.3 ns

L2 = 12 + .01 \* 100 = 13 ns

c. A two level cache hierarchy with a one cycle L1 hit time, a 20 cycle L2 hit time, a 150 cycle L2 miss penalty, a 2ns clock, a 10% L1 miss rate, and a 1% L2 miss rate.

tavg = 2 + .1 \* (40 + .01 \* 150) = 43.15 ns

L2 = 40 + .01 \* 150 = 41.5 ns

5. (40 pts) Cache Operation

a. [10] Assume a 64 Byte direct-mapped cache with one word (4B) block size. Given the sequence of memory accesses (32bit addresses) below, where each reference is a word address, convert the address to binary, then identify the tag and index of each reference. Then label each reference as a hit or a miss and show the final contents of the cache. Assume the cache is initially empty.

1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17.

Cache size: 64 B

Block size: 4 B

# blocks: 16

Associativity: Direct-mapped

# sets: 16

b. [15] Given the following 32bit addresses (as word addresses not byte addresses):

3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

Assume a three way set associative cache with two-word blocks and a total size of 24 words. Use LRU replacement. For each reference show the binary address, identify the tag and index bits and label the reference as a hit or miss, show the final contents of the cache.

c. [15] Assume the same sequence of addresses as 5b.

Assume a fully associative cache with two word blocks and a total size of 8 words. What is the miss rate if the replacement policy is LRU? What is the miss rate for MRU (most recently used)? What is the best possible miss rate for this cache given any replacement policy (the optimal policy)?

6. (20 pts) Loop Interchange for Improved Cache Performance

Given the following two nested loops, for a 1KB direct mapped cache with 16B

blocks.

a. [10] Calculate the miss ratio (misses/accesses) for each set of nested loops:

for (i=0. i < 16. i++)

for (j=0. j < 16. j++)

sum = A[i][j].

for (j = 0. j < 16. j++)

for (i =0. i < 16. i++)

sum = A[i][j].

b. [5] Explain what happens to the miss ratio for each set of loops as the size of

the arrays increases.

c. [5] What cache associativity is needed to for both sets of loops to have the

same miss ratio?