**Lab Assignment 1: Cache Memories**

Neelanjan Goswami

400414867

**Part 1: Cache Basics**

Solve the following problems and include the solution in the submitted report.

**Problem 1**

Consider a machine with a byte addressable main memory of 2^8 bytes and block size of 4 bytes. Assume that a direct mapped cache consisting of 8 lines is used with this machine.

**1) How is an 8-bit memory address divided into tag, line number, and byte number?**

Answer:

To divide an 8-bit memory address into tag, line number, and byte number for a direct-mapped cache with 8 lines and a block size of 4 bytes, you can follow these steps:

Number of Bits for Line Number = log2(Number of Lines)

Number of Lines = 8 lines

Number of Bits for Line Number = log2(8) = 3 bits

Number of Bits for Byte Number = log2(Number of Bytes in a Block)

Number of Bytes in a Block = 4 bytes

Number of Bits for Byte Number = log2(4) = 2 bits

Number of Bits for Tag = Total Address Bits - (Bits for Line Number + Bits for Byte Number)

Number of Bits for Tag = 8 bits - (3 bits + 2 bits) = 8 bits - 5 bits = 3 bits

Hence,

**- 3 bits for the tag**

**- 3 bits for the line number**

**- 2 bits for the byte number**

**2) Into what line would bytes with each of the following addresses be stored?**

0001 1011

0011 0100

1101 0000

1010 1010

Answer:

Given the 8-bit memory addresses, we can determine the cache line in which each address would be stored by looking at the line number (index) part of the address. As we determined earlier, the line number is represented by the 3 bits following the 2 offset bits in the 8-bit address. Here’s how you can determine the cache lines for each address:

000**1 10**11: The line number part is 110, which is 6 in decimal. So, this byte would be stored in cache line 6.

001**1** **01**00: The line number part is 101, which is 5 in decimal. So, this byte would be stored in cache line 5.

110**1** **00**00: The line number part is 100, which is 4 in decimal. So, this byte would be stored in cache line 4.

101**0** **10**10: The line number part is 010, which is 2 in decimal. So, this byte would be stored in cache line 2.

Hence,

0001 1011 – **Line 6**

0011 0100 – **Line 5**

1101 0000 – **Line 4**

1010 1010 – **Line 2**

**3) Suppose the byte with address 1010 0001 is stored in the cache. What are the addresses of the other bytes stored along with it?**

Answer:

In a cache memory system, a block of data from main memory is stored in a cache line. In this case, the block size is 4 bytes, which means each cache line holds 4 bytes of data.

Given the byte address 1010 0001, we know that this byte is part of a block in main memory. Since the last 2 bits of the address are used as the byte number (offset) within a block, the addresses of the other bytes in the same block will have the same values for the tag and line number parts but will have different values for the byte number part. Here are the addresses of all bytes stored along with 1010 0001 in the same block:

Byte 0: **1010 0000**

Byte 1: **1010 0001** (the original byte)

Byte 2: **1010 0010**

Byte 3: **1010 0011**

So, the addresses of the other bytes stored along with the byte at address 1010 0001 in the cache are 1010 0000, 1010 0010, and 1010 0011.

**4) How many total bytes of memory can be stored in the cache?**

Answer:

To calculate the total bytes of memory that can be stored in the cache, you can use the following formula:

Total Cache Size = Number of Lines × Block Size

Total Cache Size = 8 lines × 4 blocks

Total Cache Size = **32 bytes**

**5) Why is the tag also stored in the cache?**

Answer:

The tag is stored in the cache to uniquely identify each block of data. It helps in determining whether the desired data is in the cache (cache hit) or not (cache miss). Without the tag, we wouldn't be able to distinguish between different blocks that map to the same cache line.

**Problem 2**

Consider the following code:

cout << ”Hello World”;

cin >> a;

for(i = 0; i < 50; i++)

cout<<i;

**1) Give one example of the spatial locality in the code.**

Answer:

Spatial locality refers to the concept that when a particular memory location is accessed, the locations near it are likely to be accessed in the near future. In the context of your code, an example of spatial locality is the loop:

for(i = 0; i < 50; i++)

cout<<i;

Here, the variable **`i`** is being accessed sequentially from `0` to `49`. This is an example of spatial locality because consecutive memory locations (for each value of `i`) are being accessed in a sequence. This is beneficial for performance because modern computer systems anticipate such patterns and pre-load these nearby memory locations into the cache for faster access.

**2) Give one example of the temporal locality in the code.**

Answer:

Temporal locality refers to the concept that if a particular memory location is accessed, then it's likely to be accessed again in the near future. In the context of your code, an example of temporal locality is the repeated use of the variable `i` in the loop:

for(i = 0; i < 50; i++)

cout<<i;

Here, the variable **`i`** is being accessed multiple times - once for the comparison `i < 50`, and once for the increment `i++`, and once for printing `cout<<i`. This is an example of temporal locality because the same memory location (the one where `i` is stored) is being accessed repeatedly in a short period of time. This is beneficial for performance because modern computer systems anticipate such patterns and keep recently accessed memory locations in the cache for faster access.

**Part 2: Writing the simulator**

C++ code file (cachesim\_final.cpp), memory\_trace.txt and output.txt attached.

Command to compile the C++ file:

g++ < filename.cpp > -o < filename >

Command to run the C++ file:

./< filename> -b [ block size ] -c [ cache size ] -w [ write policy ] -a [ associativity ]

(Use ‘stride.cpp’ for simulation for all of the problems in Part 5 and 6.)

Example commands to run:  
g++ cachesim\_final.cpp -o cachesim\_final

./cachesim\_final -b 32 -c 16384 -w b -a 4 < memory\_trace.txt

Simulation output of the example:

A computer screen shot of a number

Description automatically generated

**Part 3: Spatial Locality**

For the following logic, where A and B are arrays of integer elements:

***sum=0 // sum is int***

***for(i = 0; i < num\_elements; i++)***

***sum=sum+A[i]+B[i]***

**1. Assume that A[0] is placed in address 0x0 and B[0] is in address 0x100**

**2. (Paper and Pen Analysis): Experiment with num\_elements=8, a four-way set associative cache of size 256-B and three different cache line sizes (16, 8, and 4B). For each configuration, obtain the hit rate and classify the misses, record your observations.**

Answer:

Assuming address bit length= 24 bit

Size of int = 4 bytes

cache size = 256 bytes

num\_elements=8 and a four-way set associative cache of size 256 bytes.

Cache line sizes: 16B, 8B, and 4B

Number of sets = Cache size / (Cache line size \* Number of ways)

For a four-way set associative cache, we have 4 ways.

Now, let's calculate the hit rate and classify the misses for each configuration:

**Case 1**: Cache Line Size: 4B

Number of cache lines = Cache size / Cache line size = 256B / 4B = 64 cache lines

No. of sets = 64 / 4 = 16

Bits for [ tag , index , offset ] = [ 18 , 4 , 2 ] = 24

Extra space for tag storage = 144B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 4 | 2 |
| A[0] | 000000 0000 0000 0000 | 0000 | 00 | 0 |
| A[1] | 000000 0000 0000 0000 | 0001 | 00 | 4 |
| A[2] | 000000 0000 0000 0000 | 0010 | 00 | 8 |
| A[3] | 000000 0000 0000 0000 | 0011 | 00 | 12 |
| A[4] | 000000 0000 0000 0000 | 0100 | 00 | 16 |
| A[5] | 000000 0000 0000 0000 | 0101 | 00 | 20 |
| A[6] | 000000 0000 0000 0000 | 0110 | 00 | 24 |
| A[7] | 000000 0000 0000 0000 | 0111 | 00 | 28 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 4 | 2 |
| B[0] | 000000 0000 0000 0100 | 0000 | 00 | 256 |
| B[1] | 000000 0000 0000 0100 | 0001 | 00 | 260 |
| B[2] | 000000 0000 0000 0100 | 0010 | 00 | 264 |
| B[3] | 000000 0000 0000 0100 | 0011 | 00 | 268 |
| B[4] | 000000 0000 0000 0100 | 0100 | 00 | 272 |
| B[5] | 000000 0000 0000 0100 | 0101 | 00 | 276 |
| B[6] | 000000 0000 0000 0100 | 0110 | 00 | 280 |
| B[7] | 000000 0000 0000 0100 | 0111 | 00 | 284 |

**All misses here are Compulsory Misses**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Variable** | **Operation: [Variable Address]** | **Hit/Miss** |  | **Variable** | **Operation: [Variable Address]** | **Hit/Miss** |
| A[0] | R: 0x0 | Miss |  | B[0] | R: 0x100 | Miss |
| A[1] | R: 0x4 | Miss |  | B[1] | R: 0x104 | Miss |
| A[2] | R: 0x8 | Miss |  | B[2] | R: 0x108 | Miss |
| A[3] | R: 0xC | Miss |  | B[3] | R: 0x10C | Miss |
| A[4] | R: 0x10 | Miss |  | B[4] | R: 0x110 | Miss |
| A[5] | R: 0x14 | Miss |  | B[5] | R: 0x114 | Miss |
| A[6] | R: 0x18 | Miss |  | B[6] | R: 0x118 | Miss |
| A[7] | R: 0x1C | Miss |  | B[7] | R: 0x11C | Miss |

**Total traces (read count) = 16**

**Hits = 0**

**Misses = 16 (Compulsory Misses)**

**Memory reads = 0**

**Hit rate = 0 / 16 = 0**

**Case 2**: Cache Line Size: 8B

Number of cache lines = Cache size / Cache line size = 256B / 8B = 32 cache lines

No. of sets = 32 / 4 = 8

Bits for [ tag , index , offset ] = [ 18 , 3 , 3 ] = 24

Extra space for tag storage = 72B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 3 | 3 |
| A[0] | 000000 0000 0000 0000 | 000 | 000 | 0 |
| A[1] | 000000 0000 0000 0000 | 000 | 100 | 4 |
| A[2] | 000000 0000 0000 0000 | 001 | 000 | 8 |
| A[3] | 000000 0000 0000 0000 | 001 | 100 | 12 |
| A[4] | 000000 0000 0000 0000 | 010 | 000 | 16 |
| A[5] | 000000 0000 0000 0000 | 010 | 100 | 20 |
| A[6] | 000000 0000 0000 0000 | 011 | 000 | 24 |
| A[7] | 000000 0000 0000 0000 | 011 | 100 | 28 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 3 | 3 |
| B[0] | 000000 0000 0000 0100 | 000 | 000 | 256 |
| B[1] | 000000 0000 0000 0100 | 000 | 100 | 260 |
| B[2] | 000000 0000 0000 0100 | 001 | 000 | 264 |
| B[3] | 000000 0000 0000 0100 | 001 | 100 | 268 |
| B[4] | 000000 0000 0000 0100 | 010 | 000 | 272 |
| B[5] | 000000 0000 0000 0100 | 010 | 100 | 276 |
| B[6] | 000000 0000 0000 0100 | 011 | 000 | 280 |
| B[7] | 000000 0000 0000 0100 | 011 | 100 | 284 |

**All misses here are Compulsory Misses**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Variable** | **Operation: [Variable Address]** | **Hit/Miss** |  | **Variable** | **Operation: [Variable Address]** | **Hit/Miss** |
| A[0] | R: 0x0 | Miss |  | B[0] | R: 0x100 | Miss |
| A[1] | R: 0x4 | Hit |  | B[1] | R: 0x104 | Hit |
| A[2] | R: 0x8 | Miss |  | B[2] | R: 0x108 | Miss |
| A[3] | R: 0xC | Hit |  | B[3] | R: 0x10C | Hit |
| A[4] | R: 0x10 | Miss |  | B[4] | R: 0x110 | Miss |
| A[5] | R: 0x14 | Hit |  | B[5] | R: 0x114 | Hit |
| A[6] | R: 0x18 | Miss |  | B[6] | R: 0x118 | Miss |
| A[7] | R: 0x1C | Hit |  | B[7] | R: 0x11C | Hit |

**Total traces (read count) = 16**

**Hits = 8**

**Misses = 8 (Compulsory Misses)**

**Memory reads = 8**

**Hit rate = 8 / 16 = 0.5**

**Case 3**: Cache Line Size: 16B

Number of cache lines = Cache size / Cache line size = 256B / 16B = 16 cache lines

No. of sets = 16 / 4 = 4

Bits for [ tag , index , offset ] = [ 18 , 2 , 4 ] = 24

Extra space for tag storage = 36B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 2 | 4 |
| A[0] | 000000 0000 0000 0000 | 00 | 0000 | 0 |
| A[1] | 000000 0000 0000 0000 | 00 | 0100 | 4 |
| A[2] | 000000 0000 0000 0000 | 00 | 1000 | 8 |
| A[3] | 000000 0000 0000 0000 | 00 | 1100 | 12 |
| A[4] | 000000 0000 0000 0000 | 01 | 0000 | 16 |
| A[5] | 000000 0000 0000 0000 | 01 | 0100 | 20 |
| A[6] | 000000 0000 0000 0000 | 01 | 1000 | 24 |
| A[7] | 000000 0000 0000 0000 | 01 | 1100 | 28 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 2 | 4 |
| B[0] | 000000 0000 0000 0100 | 00 | 0000 | 256 |
| B[1] | 000000 0000 0000 0100 | 00 | 0100 | 260 |
| B[2] | 000000 0000 0000 0100 | 00 | 1000 | 264 |
| B[3] | 000000 0000 0000 0100 | 00 | 1100 | 268 |
| B[4] | 000000 0000 0000 0100 | 01 | 0000 | 272 |
| B[5] | 000000 0000 0000 0100 | 01 | 0100 | 276 |
| B[6] | 000000 0000 0000 0100 | 01 | 1000 | 280 |
| B[7] | 000000 0000 0000 0100 | 01 | 1100 | 284 |

**All misses here are Compulsory Misses**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Variable** | **Operation: [Variable Address]** | **Hit/Miss** |  | **Variable** | **Operation: [Variable Address]** | **Hit/Miss** |
| A[0] | R: 0x0 | Miss |  | B[0] | R: 0x100 | Miss |
| A[1] | R: 0x4 | Hit |  | B[1] | R: 0x104 | Hit |
| A[2] | R: 0x8 | Hit |  | B[2] | R: 0x108 | Hit |
| A[3] | R: 0xC | Hit |  | B[3] | R: 0x10C | Hit |
| A[4] | R: 0x10 | Miss |  | B[4] | R: 0x110 | Miss |
| A[5] | R: 0x14 | Hit |  | B[5] | R: 0x114 | Hit |
| A[6] | R: 0x18 | Hit |  | B[6] | R: 0x118 | Hit |
| A[7] | R: 0x1C | Hit |  | B[7] | R: 0x11C | Hit |

**Total traces (read count) = 16**

**Hits = 12**

**Misses = 4 (Compulsory Misses)**

**Memory reads = 4**

**Hit rate = 12 / 16 = 0.75**

Hence, Hit rates for 4 byte cache line = 0, for 8 byte cache line = 0.5 & for 16 byte cache line = 0.75

**3. (Simulation): hand craft the traces for each of the configurations in Step 2 and run this through the simulator your created in Part 2.**

Answer:

Here are the traces for the simulation:

R:0

R:4

R:8

R:C

R:10

R:14

R:18

R:1C

R:100

R:104

R:108

R:10C

R:110

R:114

R:118

R:11C  
  
Case 1:

A screenshot of a computer

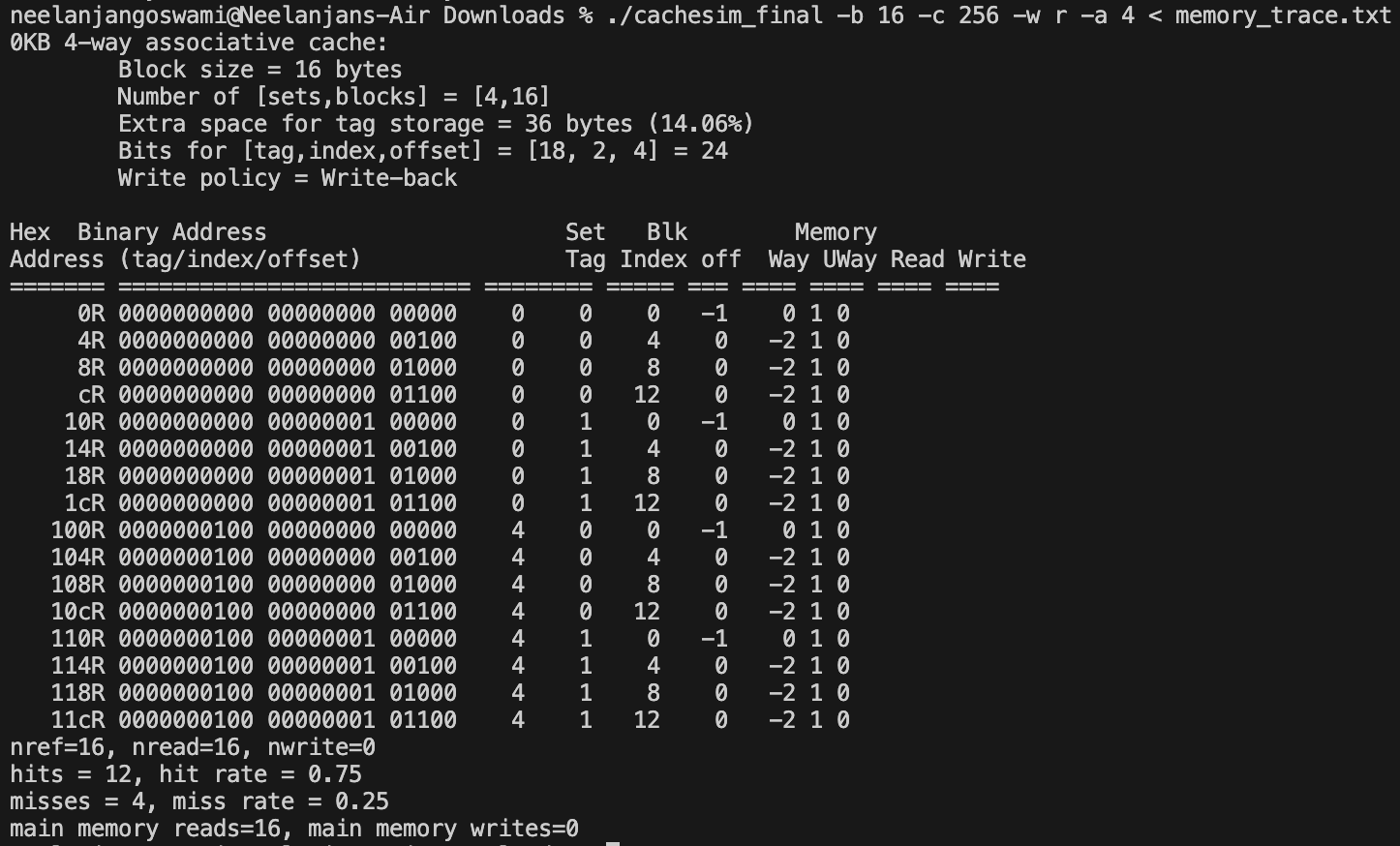
Description automatically generated

Case 2:

A screenshot of a computer

Description automatically generated

Case 3:



**4. Compare the results from step 3 and 4 and document all results.**

Answer:

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Part 4: Both Temporal and Spatial Locality**

Let’s modify the logic from part :

***sum=0***

***for (j=0; j < num\_iterations; j++)***

***for(i = 0; i < num\_elements; i++)***

***sum=sum+A[i]+B[i]***

**1. Assume that A[0] is placed in address 0x0 and B[0] is in address 0x100**

**2. (Paper and Pen Analysis): Experiment with num\_elements=8, a four-way set associative cache of size 256-B and line size of 16-B. Vary number of iterations as 1, 5, 10, and 100. For each configuration, obtain the hit rate and classify the misses, record your observations.**

Answer:  
Size of int = 4 bytes

Line size = 16 bytes  
Length of arrays in this case = num\_elements = 8

Block size = 16 bytes

Cache associativity = 4

Cache size= 256 bytes

Number of blocks = Cache lines = 256 / 16 = 16 blocks

Number of sets = 16 / 4 = 4

Assuming, Address bit length= 24 bit  
Bits for [ tag , index , offset ] = [ 18 , 2 , 4 ] = 24

Extra space for tag storage = 36 bytes

Number of iterations = 1 , 5 , 10 and 100

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 2 | 4 |
| A[0] | 000000 0000 0000 0000 | 00 | 0000 | 0 |
| A[1] | 000000 0000 0000 0000 | 00 | 0100 | 4 |
| A[2] | 000000 0000 0000 0000 | 00 | 1000 | 8 |
| A[3] | 000000 0000 0000 0000 | 00 | 1100 | 12 |
| A[4] | 000000 0000 0000 0000 | 01 | 0000 | 16 |
| A[5] | 000000 0000 0000 0000 | 01 | 0100 | 20 |
| A[6] | 000000 0000 0000 0000 | 01 | 1000 | 24 |
| A[7] | 000000 0000 0000 0000 | 01 | 1100 | 28 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 2 | 4 |
| B[0] | 000000 0000 0000 0100 | 00 | 0000 | 256 |
| B[1] | 000000 0000 0000 0100 | 00 | 0100 | 260 |
| B[2] | 000000 0000 0000 0100 | 00 | 1000 | 264 |
| B[3] | 000000 0000 0000 0100 | 00 | 1100 | 268 |
| B[4] | 000000 0000 0000 0100 | 01 | 0000 | 272 |
| B[5] | 000000 0000 0000 0100 | 01 | 0100 | 276 |
| B[6] | 000000 0000 0000 0100 | 01 | 1000 | 280 |
| B[7] | 000000 0000 0000 0100 | 01 | 1100 | 284 |

**Case 1**: Number of iterations = 1

Number of cache lines = Cache size / Cache line size = 256B / 16B = 16 cache lines

No. of sets = 16 / 4 = 4

**All misses here are Compulsory Misses**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Variable** | **Operation: [Variable Address]** | **Hit/Miss** |  | **Variable** | **Operation: [Variable Address]** | **Hit/Miss** |
| A[0] | R: 0x0 | Miss |  | B[0] | R: 0x100 | Miss |
| A[1] | R: 0x4 | Hit |  | B[1] | R: 0x104 | Hit |
| A[2] | R: 0x8 | Hit |  | B[2] | R: 0x108 | Hit |
| A[3] | R: 0xC | Hit |  | B[3] | R: 0x10C | Hit |
| A[4] | R: 0x10 | Miss |  | B[4] | R: 0x110 | Miss |
| A[5] | R: 0x14 | Hit |  | B[5] | R: 0x114 | Hit |
| A[6] | R: 0x18 | Hit |  | B[6] | R: 0x118 | Hit |
| A[7] | R: 0x1C | Hit |  | B[7] | R: 0x11C | Hit |

**Total traces (read count) = 16**

**Hits = 12**

**Misses = 4 (Compulsory Misses)**

**Memory reads = 4**

**Hit rate = 12 / 16 = 0.75**

**Case 2**: Number of iterations = 5

From the second iteration (no. of iterations 2 to 5) all the read operations are a cache hit, because in the first iteration all the memory blocks were brought into cache.

**Total traces (read count) = 80**

**Hits = 76**

**Misses = 4 (Compulsory Misses)**

**Memory reads = 4**

**Hit rate = 76 / 80 = 0.95**

**Case 3**: Number of iterations = 10

From the second iteration (no. of iterations 2 to 10) all the read operations are a cache hit, because in the first iteration all the memory blocks were brought into cache.

**Total traces (read count) = 160**

**Hits = 156**

**Misses = 4 (Compulsory Misses)**

**Memory reads = 4**

**Hit rate = 156 / 160 = 0.975**

**Case 2**: Number of iterations = 100

From the second iteration (no. of iterations 2 to 100) all the read operations are a cache hit, because in the first iteration all the memory blocks were brought into cache.

**Total traces (read count) = 1600**

**Hits = 1596**

**Misses = 4 (Compulsory Misses)**

**Memory reads = 4**

**Hit rate = 1596 / 1600 = 0.9975**

**3. (Simulation): hand craft the traces for each of the configurations in Step 2 and run this through the simulator your created in Part 2.**

Answer:

Here are the traces for the simulation:

R:0

R:4

R:8

R:C

R:10

R:14

R:18

R:1C

R:100

R:104

R:108

R:10C

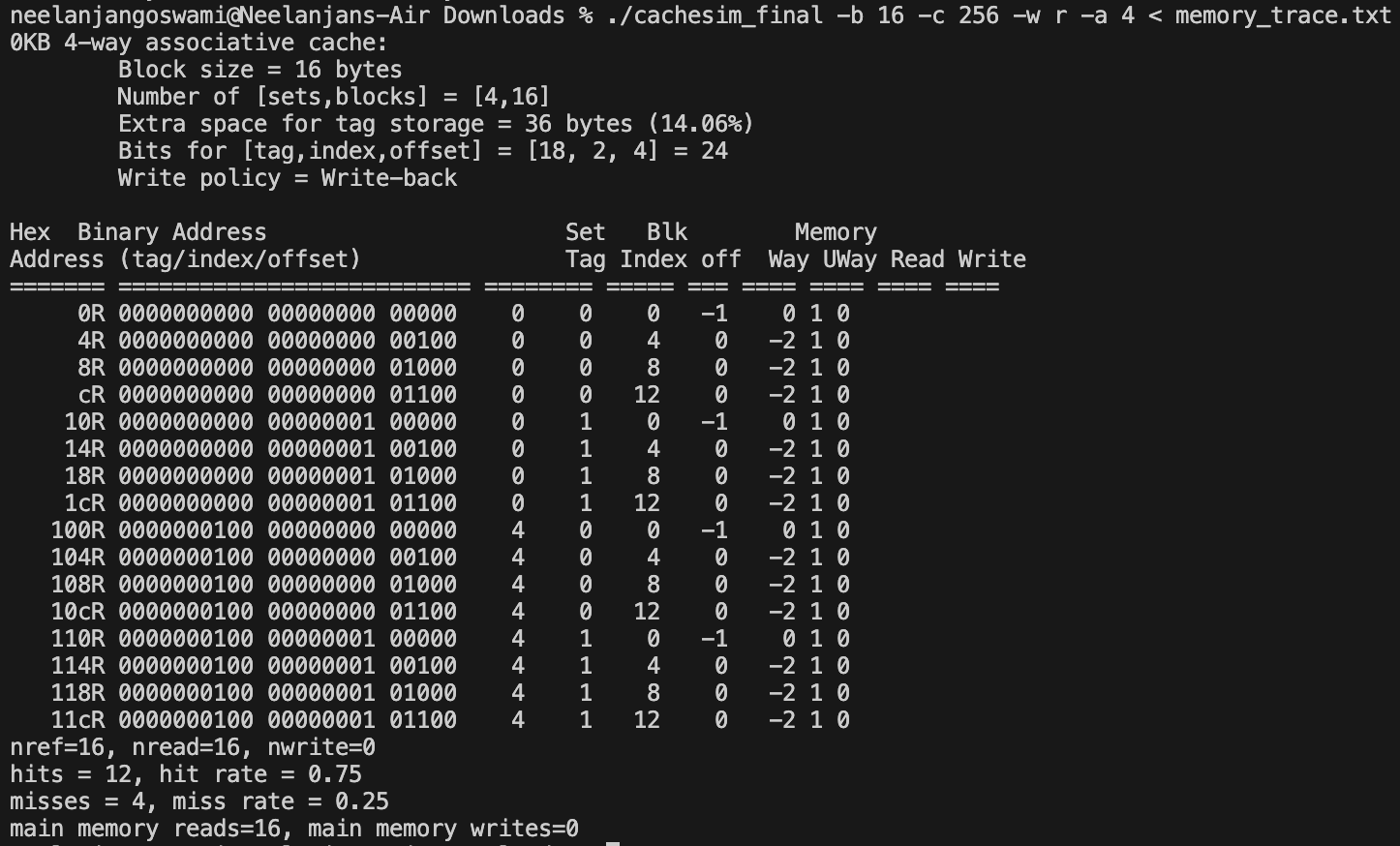
R:110

R:114

R:118

R:11C

Case 1:



Case 2:

A screen shot of a computer screen

Description automatically generated

Case 3:

A screen shot of a computer screen

Description automatically generated

Case 4:

A screen shot of a computer screen

Description automatically generated

**4. Compare the results from step 3 and 4 and document all results.**

Answer:

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**5. What you expect to change if sum and both arrays are uint\_8 instead of int?**

Answer:

When you change an integer from an int to a uint\_8, the size of the integer shrinks from 4 bytes to 1 byte. This means that the combined memory used by both arrays reduces from 32 bytes to 8 bytes, and these 8 bytes are allocated consecutively.

As the size of the int decreases, the entire array is loaded into the cache when it's first accessed. Consequently, the number of cache misses drops to just 2 compulsory misses. After that, each read request becomes a cache hit.

**Part 5: Both Temporal and Spatial Locality with Strides**

For the following logic:

***sum=0 // int***

***for (j=0; J < num\_iterations; j++)***

***for(i = 0; i < num\_elements; i=i+stride)***

***sum=sum+A[i]***

1. **Assume that A[0] is placed in address 0x0**

Use **‘stride.cpp’** for **simulation** for all of the following problems.

**2. Experiment (both paper and pen and in simulation) with num\_elements=128, a four- way set associative cache of size 256-B and line size of 16-B and number of iterations of 100. Vary the stride size as 1,2,4,8,16,21,64.**

**a. Obtain the hit rate and classify the misses**

Answer:

Size of int = 4 bytes

Line size = 16 bytes  
Length of arrays in this case = num\_elements = 8

Block size = 16 bytes

Cache associativity = 4

Cache size= 256 bytes

Number of blocks = Cache lines = 256 / 16 = 16 blocks

Number of sets = 16 / 4 = 4

Assuming, Address bit length= 24 bit  
Bits for [ tag , index , offset ] = [ 18 , 2 , 4 ] = 24

Extra space for tag storage = 36 bytes

Stride = 1 , 2 , 4 , 8 , 16 , 21 , 64

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Hex Address [tag, index, offset]** | | | **Decimal** |
| tag | index | offset |
| 18 | 2 | 4 |
| A[0] | 000000 0000 0000 0000 | 00 | 0000 | 0 |
| A[1] | 000000 0000 0000 0000 | 00 | 0100 | 4 |
| A[2] | 000000 0000 0000 0000 | 00 | 1000 | 8 |
| A[3] | 000000 0000 0000 0000 | 00 | 1100 | 12 |
| A[4] | 000000 0000 0000 0000 | 01 | 0000 | 16 |
| A[5] | 000000 0000 0000 0000 | 01 | 0100 | 20 |
| A[6] | 000000 0000 0000 0000 | 01 | 1000 | 24 |
| A[7] | 000000 0000 0000 0000 | 01 | 1100 | 28 |
| **.** | **.** | **.** | **.** | **.** |
| **.** | **.** | **.** | **.** | **.** |
| **.** | **.** | **.** | **.** | **.** |
| A[124] | 000000 0000 0000 0111 | 11 | 0000 | 496 |
| A[125] | 000000 0000 0000 0111 | 11 | 0100 | 500 |
| A[126] | 000000 0000 0000 0111 | 11 | 1000 | 504 |
| A[127] | 000000 0000 0000 0111 | 11 | 1100 | 508 |

**Case 1**: Stride = 1

Value of i = 0 , 1 , 2 , 3 , ... , 125 , 126 ,127

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[4...7] | A[8...11] | A[12...15] |
| 01 | A[16...19] | A[20...23] | A[24...27] | A[28...31] |
| 10 | A[32...35] | A[36...39] | A[40...43] | A[44...47] |
| 11 | A[48...51] | A[52...55] | A[56...59] | A[60...63] |

Therefore, in 1 loop:

Total Traces (read count) = 128  
Misses = 32 misses (compulsory/cold miss)

Hits = 96

**Number of iterations = 100**

**Total Read Count = 12,800**

**Cache Misses = 3200 :-**

**Cold Misses = 32**

**Capacity Misses = 3168**

**Hits = 9600**

**Hit rate = 9600/12800 = 0.75**

**A black screen with white text

Description automatically generated**

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 2**: Stride = 2

Value of i = 0 , 2 , 4 , ... , 124 , 126

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[16...19] | A[32...35] | A[48...51] |
| 01 | A[4...7] | A[20...23] | A[36...39] | A[52...55] |
| 10 | A[8...11] | A[24...27] | A[40...43] | A[56...59] |
| 11 | A[12...15] | A[28...31] | A[44...47] | A[60...63] |

Therefore, in 1 loop:

Total Traces (read count) = 64  
Misses = 32 misses (compulsory/cold miss)

Hits = 32

**Number of iterations = 100**

**Total Read Count = 6400**

**Cache Misses = 3200 :-**

**Cold Misses = 32**

**Capacity Misses = 3168**

**Hits = 3200**

**Hit rate = 3200/6400 = 0.5**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 3**: Stride = 4

Value of i = 0 , 4 , 8 , 12 , ... , 120 , 124

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[16...19] | A[32...35] | A[48...51] |
| 01 | A[4...7] | A[20...23] | A[36...39] | A[52...55] |
| 10 | A[8...11] | A[24...27] | A[40...43] | A[56...59] |
| 11 | A[12...15] | A[28...31] | A[44...47] | A[60...63] |

Therefore, in 1 loop:

Total Traces (read count) = 32  
Misses = 32 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 100**

**Total Read Count = 3200**

**Cache Misses = 3200 :-**

**Cold Misses = 32**

**Capacity Misses = 3168**

**Hits = 0**

**Hit rate = 0/3200 = 0**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 4**: Stride = 8

Value of i = 0 , 8 , 16 , ... , 116 , 120

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[16...19] | A[32...35] | A[48...51] |
| 01 |  |  |  |  |
| 10 | A[8...11] | A[24...27] | A[40...43] | A[56...59] |
| 11 |  |  |  |  |

Therefore, in 1 loop:

Total Traces (read count) = 16  
Misses = 16 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 100**

**Total Read Count = 1600**

**Cache Misses = 1600 :-**

**Cold Misses = 16**

**Capacity Misses = 1584**

**Hits = 0**

**Hit rate = 0/1600 = 0**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 5**: Stride = 16

Value of i = 0 , 16 , 32 , ... , 96 , 112

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[16...19] | A[32...35] | A[48...51] |
| 01 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |

Therefore, in 1 loop:

Total Traces (read count) = 8  
Misses = 8 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 100**

**Total Read Count = 800**

**Cache Misses = 800 :-**

**Cold Misses = 8**

**Capacity Misses = 792**

**Hits = 0**

**Hit rate = 0/800 = 0**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 6**: Stride = 21

Value of i = 0 , 21 , 42 , 63 , 84 , 105 , 126

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] |  |  |  |
| 01 | A[20...23] | A[84...87] |  |  |
| 10 | A[40...43] | A[104...107] |  |  |
| 11 | A[60...63] | A[124...127] |  |  |

Therefore, in 1 loop:

Total Traces (read count) = 7  
Misses = 7 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 100**

**Total Read Count = 700**

**Cache Misses = 99 :-**

**Cold Misses = 99**

**Hits = 601**

**Hit rate = 693/700 = 0.14**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 7**: Stride = 64

Value of i = 0 , 64

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[64…67] |  |  |
| 01 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |

Therefore, in 1 loop:

Total Traces (read count) = 2  
Misses = 2 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 100**

**Total Read Count = 200**

**Cache Misses = 200 :-**

**Cold Misses = 200**

**Hits = 0**

**Hit rate = 0/200 = 0**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**3. Repeat Step 2 for number of iterations=10 and 50**

Answer:

Initial steps same as the previous answer.

**Case 1**: Stride = 1

Value of i = 0 , 1 , 2 , 3 , ... , 125 , 126 ,127

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[4...7] | A[8...11] | A[12...15] |
| 01 | A[16...19] | A[20...23] | A[24...27] | A[28...31] |
| 10 | A[32...35] | A[36...39] | A[40...43] | A[44...47] |
| 11 | A[48...51] | A[52...55] | A[56...59] | A[60...63] |

Therefore, in 1 loop:

Total Traces (read count) = 128  
Misses = 32 misses (compulsory/cold miss)

Hits = 96

**Number of iterations = 10**

**Total Read Count = 1280**

**Cache Misses = 320 :-**

**Cold Misses = 32**

**Capacity Misses = 288**

**Hits = 960**

**Hit rate = 960/1280 = 0.75**

**A black screen with white text

Description automatically generated**

**Number of iterations = 50**

**Total Read Count = 6400**

**Cache Misses = 1600 :-**

**Cold Misses = 32**

**Capacity Misses = 1568**

**Hits = 4800**

**Hit rate = 4800/6400 = 0.75**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 2**: Stride = 2

Value of i = 0 , 2 , 4 , ... , 124 , 126

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[16...19] | A[32...35] | A[48...51] |
| 01 | A[4...7] | A[20...23] | A[36...39] | A[52...55] |
| 10 | A[8...11] | A[24...27] | A[40...43] | A[56...59] |
| 11 | A[12...15] | A[28...31] | A[44...47] | A[60...63] |

Therefore, in 1 loop:

Total Traces (read count) = 64  
Misses = 32 misses (compulsory/cold miss)

Hits = 32

**Number of iterations = 10**

**Total Read Count = 640**

**Cache Misses = 320 :-**

**Cold Misses = 32**

**Capacity Misses = 288**

**Hits = 320**

**Hit rate = 320/640 = 0.5**

A black screen with white text

Description automatically generated

**Number of iterations = 50**

**Total Read Count = 3200**

**Cache Misses = 1600 :-**

**Cold Misses = 32**

**Capacity Misses = 1568**

**Hits = 1600**

**Hit rate = 1600/3200 = 0.5**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 3**: Stride = 4

Value of i = 0 , 4 , 8 , 12 , ... , 120 , 124

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[16...19] | A[32...35] | A[48...51] |
| 01 | A[4...7] | A[20...23] | A[36...39] | A[52...55] |
| 10 | A[8...11] | A[24...27] | A[40...43] | A[56...59] |
| 11 | A[12...15] | A[28...31] | A[44...47] | A[60...63] |

Therefore, in 1 loop:

Total Traces (read count) = 32  
Misses = 32 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 10**

**Total Read Count = 320**

**Cache Misses = 320 :-**

**Cold Misses = 32**

**Capacity Misses = 288**

**Hits = 0**

**Hit rate = 0/320 = 0**

A black screen with white text

Description automatically generated

**Number of iterations = 50**

**Total Read Count = 1600**

**Cache Misses = 1600 :-**

**Cold Misses = 32**

**Capacity Misses = 1568**

**Hits = 0**

**Hit rate = 0/1600 = 0**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 4**: Stride = 8

Value of i = 0 , 8 , 16 , ... , 116 , 120

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[16...19] | A[32...35] | A[48...51] |
| 01 |  |  |  |  |
| 10 | A[8...11] | A[24...27] | A[40...43] | A[56...59] |
| 11 |  |  |  |  |

Therefore, in 1 loop:

Total Traces (read count) = 16  
Misses = 16 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 10**

**Total Read Count = 160**

**Cache Misses = 160 :-**

**Cold Misses = 16**

**Capacity Misses = 144**

**Hits = 0**

**Hit rate = 0/160 = 0**

A black screen with white text

Description automatically generated

**Number of iterations = 50**

**Total Read Count = 800**

**Cache Misses = 800 :-**

**Cold Misses = 16**

**Capacity Misses = 784**

**Hits = 0**

**Hit rate = 0/800 = 0**

A screen shot of a computer

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 5**: Stride = 16

Value of i = 0 , 16 , 32 , ... , 96 , 112

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[16...19] | A[32...35] | A[48...51] |
| 01 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |

Therefore, in 1 loop:

Total Traces (read count) = 8  
Misses = 8 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 10**

**Total Read Count = 80**

**Cache Misses = 80 :-**

**Cold Misses = 8**

**Capacity Misses = 72**

**Hits = 0**

**Hit rate = 0/80 = 0**

A black screen with white text

Description automatically generated

**Number of iterations = 50**

**Total Read Count = 400**

**Cache Misses = 400 :-**

**Cold Misses = 8**

**Capacity Misses = 392**

**Hits = 0**

**Hit rate = 0/400 = 0**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 6**: Stride = 21

Value of i = 0 , 21 , 42 , 63 , 84 , 105 , 126

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] |  |  |  |
| 01 | A[20...23] | A[84...87] |  |  |
| 10 | A[40...43] | A[104...107] |  |  |
| 11 | A[60...63] | A[124...127] |  |  |

Therefore, in 1 loop:

Total Traces (read count) = 7  
Misses = 7 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 10**

**Total Read Count = 70**

**Cache Misses = 7 :-**

**Cold Misses = 7**

**Hits = 63**

**Hit rate = 63/70 = 0.9**

A black screen with white text

Description automatically generated

**Number of iterations = 50**

**Total Read Count = 350**

**Cache Misses = 7 :-**

**Cold Misses = 7**

**Hits = 343**

**Hit rate = 343/350 = 0.98**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**Case 7**: Stride = 64

Value of i = 0 , 64

Cache is filed in this process:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Way 00** | **Way 01** | **Way 10** | **Way 11** |
| 00 | A[0...3] | A[64…67] |  |  |
| 01 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |

Therefore, in 1 loop:

Total Traces (read count) = 2  
Misses = 2 misses (compulsory/cold miss)

Hits = 0

**Number of iterations = 10**

**Total Read Count = 20**

**Cache Misses = 20 :-**

**Cold Misses = 20**

**Hits = 0**

**Hit rate = 0/20 = 0**

A black screen with white text

Description automatically generated

**Number of iterations = 50**

**Total Read Count = 200**

**Cache Misses = 200 :-**

**Cold Misses = 2**

**Hits = 0**

**Hit rate = 0/200 = 0**

A black screen with white text

Description automatically generated

Both pen and paper analysis and simulator results **match, they correspond with each other.**

**4. How does varying number of iterations and stride value affect the hit rate? How does this relate to spatial and temporal locality?**

Answer:

When you increase the stride value, initially it negatively impacts the hit rate, causing it to decrease. Once it becomes equal to the size of an integer, the hit rate drops to zero. When you continue to increase the stride in multiples of 2, the hit rate remains at zero because all the accessed elements are mapped to the same cache set. On the other hand, if you increase the stride so much that the cache isn't fully filled, after the first iteration, it's always a hit because there's no conflict. As you increase the number of iterations, hits generally increase because there are no cold misses. However, this outcome depends significantly on the stride value. In cases of spatial locality, a smaller stride value increases the cache hit rate, and vice versa. This happens because the entire block is prefetched, and adjacent memory locations are accessed quickly.

In terms of temporal locality, the cache status is crucial. I observed that when the stride value is chosen in a way that the memory addresses are mapped to the same cache set, there are very few cache hits. This is because there's a conflict miss every time. The best scenario occurs when the stride is either too large or not a power of 2.

**Part 6: Cache Architecture and Replacement Policies**

**1. For the same logic in Part 5, experiment (both paper and pen and in simulation) with num\_elements=128, a cache size of size 256-B and line size of 16-B, a stride of 1 and**

**number of iterations of 100. Obtain the hit rate and identify the final cache content for the following cache architectures**

1. **Direct Mapped**

Answer:

Length of array = num\_elements = 128

Cache size = 256 bytes

Block size = 16 bytes

Stride = 1

No. of blocks = 256 / 16 = 16

Num\_iterations = 100

Associativity = 1

No. of sets = 16

[ Tag , Index , Offset ] = [ 16 , 4 , 4 ] = 24 bits

After number of iterations = 100, cache state:

|  |  |
| --- | --- |
| **Index** | **Value** |
| 0000 | A[64, 65, 66, 67] |
| 0001 | A[68, 69, 70, 71] |
| **.** | **.** |
| **.** | **.** |
| **.** | **.** |
| 1110 | A[120, 121, 122, 123] |
| 1111 | A[124, 125, 126, 127] |

In one loop:

**Total traces (read count) = 128**

**Cache misses = 32 (compulsory/cold miss)**

**Hits = 96**

Therefore,

**For number of iterations = 100:**

**Total count = 12800**

**Cache misses = 3200**

**Cold misses = 32**

**Hit = 9600**

**Hit rate = 9600/12800 = 0.75**

**A black screen with white text

Description automatically generated**

Both pen and paper analysis and simulator results **match, they correspond with each other.**

1. **2-way Set Associative**

Answer:

Initial steps same as previous answer.

Associativity = 2

No. of sets = 16 / 2 = 8

[ Tag , Index , Offset ] = [ 17 , 3 , 4 ] = 24 bits

After number of iterations = 100, cache state:

|  |  |  |
| --- | --- | --- |
| **Index** | **Value ( Way 0 )** | **Value ( Way 1 )** |
| 000 | A[64, 65, 66, 67] | A[96, 97, 98, 99] |
| 001 | A[68, 69, 70, 71] | A[100, 101, 102, 103] |
| **.** | **.** | **.** |
| **.** | **.** | **.** |
| **.** | **.** | **.** |
| 110 | A[88, 89, 90, 91] | A[120, 121, 122, 123] |
| 111 | A[92, 93, 94, 95] | A[124, 125, 126, 127] |

In one loop:

**Total traces (read count) = 128**

**Cache misses = 32 (compulsory/cold miss)**

**Hits = 96**

Therefore,

**For number of iterations = 100:**

**Total count = 12800**

**Cache misses = 3200**

**Cold misses = 32**

**Capacity misses = 3168**

**Hit = 9600**

**Hit rate = 9600/12800 = 0.75**

Both pen and paper analysis and simulator results **match, they correspond with each other.**

1. **4-way Set Associative**

Answer:

Initial steps same as previous answer.

Associativity = 4

No. of sets = 16 / 2 = 4

[ Tag , Index , Offset ] = [ 18 , 2 , 4 ] = 24 bits

After number of iterations = 100, cache state:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Value ( Way 00 )** | **Value ( Way 01 )** | **Value ( Way 10 )** | **Value ( Way 11 )** |
| 00 | A[64, 65, 66, 67] | A[80, 81, 82, 83] | A[96, 97, 98, 99] | A[112, 113, 114, 115] |
| 01 | A[68, 69, 70, 71] | A[84, 85, 86, 87] | A[100, 101, 102, 103] | A[116, 117, 118, 119] |
| 10 | A[72, 73, 74, 75] | A[88, 89, 90, 91] | A[104, 105, 106, 107] | A[120, 121, 122, 123] |
| 11 | A[76, 77, 78, 79] | A[92, 93, 94, 95] | A[108, 109, 110, 111] | A[124, 125, 126, 127] |

In one loop:

**Total traces (read count) = 128**

**Cache misses = 32 (compulsory/cold miss)**

**Hits = 96**

Therefore,

**For number of iterations = 100:**

**Total count = 12800**

**Cache misses = 3200**

**Cold misses = 32**

**Capacity misses = 3168**

**Hit = 9600**

**Hit rate = 9600/12800 = 0.75**

**A black screen with white text

Description automatically generated**

Both pen and paper analysis and simulator results **match, they correspond with each other.**

1. **Fully-Associative Cache**

Answer:

Number of blocks = 16

[ Tag , Offset ] = [ 20 , 4 ] = 24 bits

After number of iterations = 100, cache state:

|  |  |
| --- | --- |
| **Block** | **Value** |
| 1 | A[64, 65, 66, 67] |
| 2 | A[68, 69, 70, 71] |
| **.** | **.** |
| **.** | **.** |
| **.** | **.** |
| 15 | A[120, 121, 122, 123] |
| 16 | A[124, 125, 126, 127] |

In one loop:

**Total traces (read count) = 128**

**Cache misses = 32 (Compulsory / Cold misses )**

**Hits = 96**

Therefore,

**For number of iterations = 100:**

**Total count = 12800**

**Cache misses = 3200**

**Cold misses = 32**

**Capacity misses = 3168**

**Hit = 9600**

**Hit rate = 9600/12800 = 0.75**

**A black screen with white text

Description automatically generated**

Both pen and paper analysis and simulator results **match, they correspond with each other.**