CSCE 2303 - Dr Mohamed Shalan

Computer Org. and Assembly Language
Project 2 Report

Nour Kasaby - 900211955
Nadine Safwat - 900212508
Mariam ElGhobary - 900211608
Youssef Rami - 900211812

Project Outline

For this project we were asked to create an n-way set-associative cache simulator that will accept given memory addresses in the form of integers, we are given a set size of 16 Kbytes for the cache and 64 Mbytes for the memory, and, for a variable number of ways and line sizes we are asked to calculate the hit ratio for each given memory address generator. Furthermore we are required to create test cases in order to verify our code's functionality.

Test Cases

Number of ways = 4 ; Line Size = 32

100 Integers

For the first test case, we created a memory address generator that increments by 4 with every iteration in order to mimic the reading of integers from an array. With the assumption that the first element is mapped to the first byte of the first cache block we can calculate the expected hit ratio that will result from reading 100 Integers.

$\frac{32}{4} = 8\ integers/line$

$8 \times 4 = 32\ integers/set$

$\frac{100}{32} = 3.125 \rightarrow 3\ sets\ and\ 1\ line\ are\ filled$

$(3 \times 4) + 1 = 13\ misses$

$\frac{100-13}{100} = 0.87 = expected\ hit\ ratio$

$Actual\ hit\ ratio = 0.87$

1000 Characters

Similarly to the second test case, we created a memory address generator that increments by 1 with every interaction to mimic the reading of characters from an array. Using the same assumptions and calculations we get the expected hit ratio

$\frac{32}{1} = 32\ characters/line$

$32 \times 4 = 128\ characters/set$

$\frac{1000}{128} = 7.8125 \rightarrow 7\ sets\ and\ 4\ lines\ are\ filled$

$(7 \times 4) + 4 = 32\ misses$

$\frac{1000-32}{1000} = 0.968 = expected\ hit\ ratio$

$Actual\ hit\ ratio = 0.968$

Data Analysis

In order to understand the trends seen in hit ratios we must calculate the expected hit ratio by analyzing the behavior of each memory address generator and understanding if the changing factors will have any effect. Afterwards all calculated hit ratios can be compared to the results obtained by running the program to validate its functionality

Experiment 1

Overview:

Setting the number of ways to 4, we test line sizes, 16, 32, 64 and 128 bytes and calculate the hit ratio for all 6 generators after a million iterations

MemGen1

This is a sequential memory address generator therefore the number of misses is equal to the number of blocks used.

Line size: 16

$$\frac{1{,}000{,}000}{16} = 62500 = number\ of\ blocks\ used = misses$$

$$\frac{1{,}000{,}000 - 62500}{1{,}000{,}000} = 0.9375 = expected\ hit\ ratio$$

Line size: 32

$$\frac{1{,}000{,}000}{32} = 31250 = number\ of\ blocks\ used = misses$$

$$\frac{1{,}000{,}000 - 31250}{1{,}000{,}000} = 0.96875 = expected\ hit\ ratio$$

Line size: 64

$$\frac{1{,}000{,}000}{64} = 15625 = number\ of\ blocks\ used = misses$$

$$\frac{1{,}000{,}000 - 15625}{1{,}000{,}000} = 0.984375 = expected\ hit\ ratio$$

Line size: 128

$$\frac{1{,}000{,}000}{128} = 7812.5 = number\ of\ blocks\ used = misses$$

$$\frac{1{,}000{,}000 - 7813}{1{,}000{,}000} = 0.992187 = expected\ hit\ ratio$$

MemGen2

This function makes use of the given random number generator to create the memory addresses, however we modulo this number by $(24 \times 1024) = 24576$ which means that the only addresses that can be generated by the function range from 0 to 24575. Using this information, along with the given size of the cache $(16 \times 1024) = 16384$ we can calculate the probability of referencing each address by the cache

$$\frac{(16\times1024)}{(24\times1024)} = \frac{2}{3} \simeq 0.666667$$

This hit ratio is applicable to all the line sizes as it only takes the size of the whole cache into account.

MemGen3

This function makes use of the given random number generator to create the memory addresses, however we modulo this number by $(64 \times 1024 \times 1024) = 67108864$ which means that the only addresses that can be generated by the function range from 0 to 67108863. Using this information, along with the given size of the cache $(16 \times 1024) = 16384$ we can calculate the probability of referencing each address by the cache

$$\frac{(16\times1024)}{(64\times1024\times1024)} = \frac{1}{4096} \simeq 0.000244$$

This hit ratio is applicable to all the line sizes as it only takes the size of the whole cache into account.

MemGen4

This function will increment the address sequentially but, due to the modulo $(4 \times 1024) = 4096$ the largest address that results is 4095.

$\frac{1000000}{4096} \simeq 244$, meaning addresses 0 to 4095 will be generated sequentially 244 times but because the number of addresses is less than the cache size, the cache will not be fully filled, and there will only be misses when the lines are initially filled. Therefore, for each line size we will calculate the number of lines used and obtain from it the expected hit ratio.

Line size: 16

$$\frac{4096}{16} = 256 = number\ of\ blocks\ used = misses$$
$$\frac{1000000-256}{1000000} = 0.999744 = expected\ hit\ ratio$$

Line size: 32

$$\frac{4096}{32} = 128 = number\ of\ blocks\ used = misses$$
$$\frac{1000000-128}{1000000} = 0.999872 = expected\ hit\ ratio$$

Line size: 64

$$\frac{4096}{64} = 64 = number\ of\ blocks\ used = misses$$
$$\frac{1000000-64}{1000000} = 0.999936 = expected\ hit\ ratio$$

Line size: 128

$$\frac{4096}{128} = 32 = number\ of\ blocks\ used = misses$$
$$\frac{1000000-32}{1000000} = 0.999968 = expected\ hit\ ratio$$

MemGen5

This function is also a sequential generator; however, we apply modulus 65536 meaning the addresses we can obtain are 0-65535. Seeing as the cache size is only 16384 bytes we will initially fill the cache with the first 16384 addresses and the generator will behave in the same way as memGen1 due to the capacity being filled. Therefore, we can conclude that the expected hit ratios for this generator are equal to the expected hit ratios of memGen1.

MemGen6

In this function, we increment the address by 32 bytes with every iteration. This means that:

Line size: 16

Every iteration will result in a miss as there is, effectively, no spatial locality.

Line size: 32

Same as the 16 byte line, spatial locality does not hold for this case and every iteration results in a miss

Line size: 64

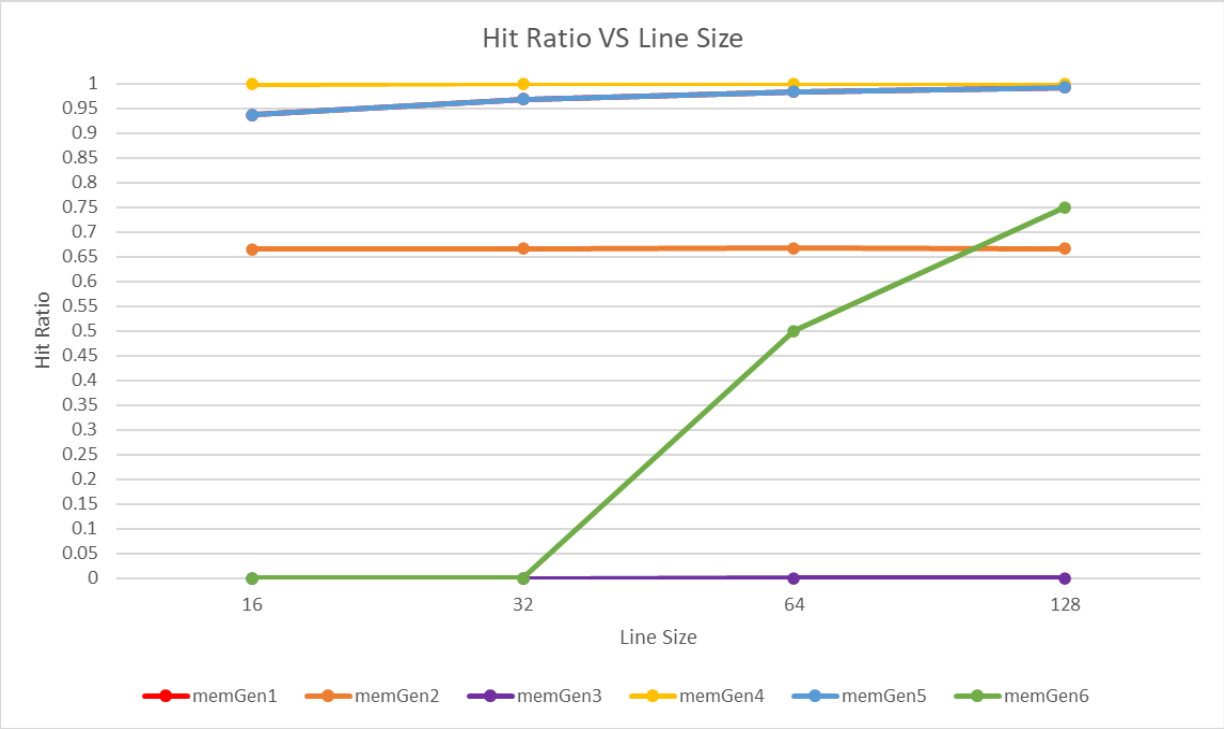
Spatial locality is only partially effective in this case as it will always be one miss and one hit. This means that the expected hit ratio is approximately 0.5

Line size: 128

In the same sense as the 64 byte line, the sequence will always be one miss and three hits which means that the expected hit ratio is $\frac{3}{4} = 0.75$

Results:

		Line Size			
		16	**32**	**64**	**128**
Memory Generator	**memGen1**	0.9375	0.96875	0.984375	0.992187
	memGen2	0.665758	0.666452	0.66742	0.666517
	memGen3	0.000253	0.00022	0.000231	0.000243
	memGen4	0.999744	0.999872	0.999936	0.999968
	memGen5	0.9375	0.96875	0.984375	0.992187
	memGen6	0	0	0.499999	0.749999



NOTE: memGen1 is hidden behind memGen5

Experiment 2

Overview

Setting the line size to 32 bytes, we test associativities, 1, 2, 4, 8, 16 and calculate the hit ratio for all 6 generators after a million iterations. However, if we notice from the calculations done to validify the results seen in experiment 1, it is obvious that the number of ways does not affect the hit ratio, therefore all generators will have the same hit ratio of experiment 1 with line size 32 bytes.

To understand why the number of ways has no effect in this case, we will first calculate the number of sets for every associativity:

Associativity: 1

$$\frac{(16\times1024)}{32\times1} = 512\ sets$$

Associativity: 2

$$\frac{(16\times1024)}{32\times1} = 256\ sets$$

Associativity: 4

$$\frac{(16\times1024)}{32\times1} = 128\ sets$$

Associativity: 8

$$\frac{(16\times1024)}{32\times1} = 64\ sets$$

Associativity: 16

$$\frac{(16\times1024)}{32\times1} = 32\ sets$$

MemGen1

There is no temporal locality in memGen1 as the generator is sequential and this is why changing the number of ways does not have an effect on the hit ratio. Therefore, we can use the same calculations we did in experiment 1.

Line size 32:

$$\frac{1{,}000{,}000}{32} = 31250 = number\ of\ blocks\ used = misses$$

$$\frac{1{,}000{,}000 - 31250}{1{,}000{,}000} = 0.96875 = expected\ hit\ ratio$$

MemGen2

As stated in the data analysis of experiment 1 for this generator, the expected hit ratio we calculated does not rely on either line size or associativity, only cache size which is common in both experiments

$$\frac{(16\times1024)}{(24\times1024)} = \frac{2}{3} \simeq 0.666667$$

MemGen3

As stated in the data analysis of experiment 1 for this generator, the expected hit ratio we calculated does not rely on either line size or associativity, only cache size which is common in both experiments

$$\frac{(16\times1024)}{(64\times1024\times1024)} = \frac{1}{4096} \simeq 0.000244$$

MemGen4

We have, for every associativity, that the number of addresses that are produced (4096) will be less than the available space so the only misses that happen are the initial cold start misses for every line.

Line size: 32

$$\frac{4096}{32} = 128 = number\ of\ blocks\ used = misses$$

$$\frac{1000000-128}{1000000} = 0.999872 = expected\ hit\ ratio$$

MemGen5

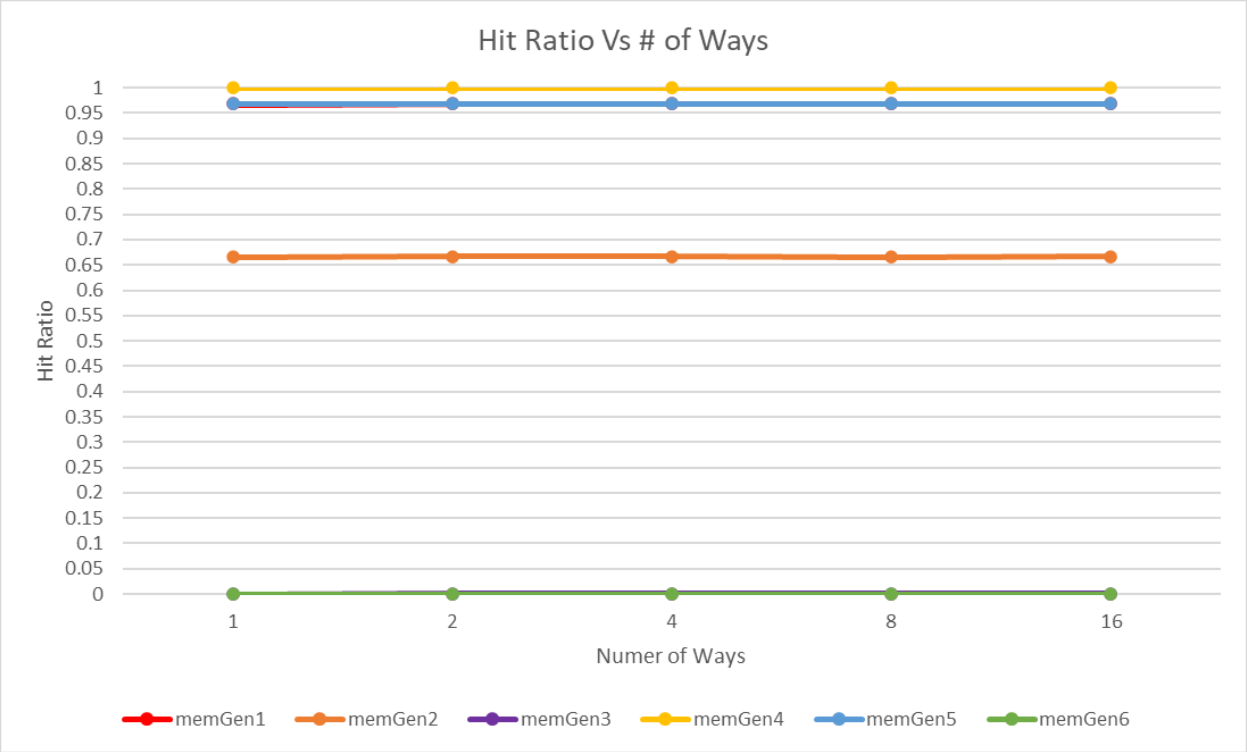
As we stated before, memGen5 adapts a very similar pattern as memGen1 which results in them having the same expected hit ratios. The same reasoning applies in this experiment as well.

MemGen6

Because this generator jumps 32 bits per iteration, the associativity has no effect on the performance of the cache as spatial locality is nullified. The hit ratio will always be 0

Results:

		# of Ways				
		1	**2**	**4**	**8**	**16**
Memory Generator	**memGen1**	0.9687	0.96875	0.96875	0.96875	0.96875
	memGen2	0.666057	0.666539	0.666808	0.665615	0.66669
	memGen3	0.000228	0.000235	0.000243	0.000247	0.000265
	memGen4	0.999872	0.999872	0.999872	0.999872	0.999872
	memGen5	0.96875	0.96875	0.96875	0.96875	0.96875
	memGen6	0	0	0	0	0



NOTE: memGen1 is hidden behind memGen5 & memGen3 is hidden behind memGen6

Conclusions:

Calculated hit ratios are corroborated by the results found in the experiment which proves the validity of our program. It is also important to note that all of the results for line size 32 in experiment 1 match the results found in experiment 2 as the associativity seemed to have no effect on any of the hit ratios of any of the generators. Reasons for this are explained above in the data analysis section.

Some generators work better than others as they have more spatial or temporal locality. For example memGen4 works better than memGen1 due to the repeating of a memory address that does not exist in memGen1 (temporal locality) or memGen1 working better than memGen6 as it takes advantage of spatial locality where as memGen6 jumps further in addresses.