CS224
Section No.: 6
Spring 2020
Lab No.: 6
Name / Bilkent ID: Osama Tanveer / 21801147

PART 1

1.

No.	Cache Size KB	N way cache	Word Size in bits	Block Size (no. of words)	No. of Sets	Tag Size in bits	Index Size (Set No.) in bits	Word Block Offset Size in bits	Byte Offset Size in bits	Block Replacement Policy Needed
1	**2**	**1**	**32**	**4**	2^7	18	7	2	2	No
2	**2**	**2**	**32**	**4**	2^6	19	6	2	2	Yes
3	**2**	**4**	**32**	**8**	2^4	20	4	3	2	Yes
4	**2**	**Full**	**32**	**8**	1	24	0	3	2	Yes
9	**16**	**1**	**16**	**4**	2^{11}	15	11	2	1	No
10	**16**	**2**	**16**	**4**	2^{10}	16	10	2	1	Yes
11	**16**	**4**	**8**	**16**	2^8	17	8	4	0	Yes
12	**16**	**Full**	**8**	**16**	1	25	0	4	0	Yes

2.

a.

Instruction	**Iteration No.**				
	1	**2**	**3**	**4**	**5**
lw $t1, 0xA4($t0)	Compulsory Miss	Hit	Hit	Hit	Hit
lw $t2, 0xAC($0)	Compulsory Miss	Hit	Hit	Hit	Hit
lw $t3, 0xA8($0)	Hit	Hit	Hit	Hit	Hit

b.

Main memory size = 2^{32} bits
Instruction Length = $\log_2 (2^{32})$ = 32 bits
Cache is direct mapped because N = 1 and has 8 words.
Number of blocks in cache = 8/2 = 4 blocks

N is 1, so Number of sets = 4
1 word = 2^2 bytes
Byte offset = $\log_2(2^2) = 2$
Number of words in a block = 2
Block offset = $\log_2(2) = 1$
Index size = $\log_2(2^2) = 2$
Tag size = 32 - 2 - 1 - 2 = 27

Every set in the cache contains 1 valid bit, 27 tag bits and 64 data bits.
Number of bits in one set in cache = 1 + 27 + 64 = 92 bits
Cache Size = 92 bits $\times$ 4 = 368 bits

The cache size is 368 bits.

c.
2 comparators are needed to compare sets.
2 AND gates are needed to check the valid bit and the output from the comparator.
1 OR gate is needed to show the hit from either of the 2 blocks.
1 32 bit 2-to-1 mux is needed to select the word output.

3.

a.

Instruction	**Iteration No.**				
	1	**2**	**3**	**4**	**5**
lw $t1, 0xA4($t0)	Compulsory Miss	Capacity Miss	Capacity Miss	Capacity Miss	Capacity Miss
lw $t2, 0xAC($0)	Compulsory Miss	Capacity Miss	Capacity Miss	Capacity Miss	Capacity Miss
lw $t3, 0xA8($0)	Capacity Miss	Capacity Miss	Capacity Miss	Capacity Miss	Capacity Miss

b.
Main memory size = 2^{32} bits
Instruction Length = $\log_2(2^{32}) = 32$ bits
Cache is 2 way associative because N = 2 and has 2 words because each block has 1 word.
Number of blocks in cache = 2/1 = 2 blocks
N is 2, so Number of sets = 1
1 word = 2^2 bytes
Byte offset = $\log_2(2^2) = 2$
Number of words in a block = 1
Block offset = $\log_2(1) = 0$

Index size = $\log_2(1) = 0$
Tag size = 32 - 2 - 0 - 0 = 30

Every set in the cache contains 1 use bit, and 2 blocks.
In each block:

- 1 valid bit
- 30 tag bits
- 32 data bits

Cache Size = $1 \times (1 + 2 \times (1 + 30 + 32)) = 127$ bits

c.
2 comparators needed to compare sets.
2 AND gates needed to check the valid bit and output of the comparator.
1 32 bit 2-to-1 mux to select output word.
1 OR gate to show hit from a block.

4. **Assembly program for matrices observations.**

```
        .text
.globl __main

__main:
        # Menu display
        la $a0, menuOp1
        li $v0, 4
        syscall

        la $a0, menuOp2
        li $v0, 4
        syscall

        la $a0, menuOp3
        li $v0, 4
        syscall

        la $a0, menuOp4
        li $v0, 4
        syscall

        la $a0, menuOp5
        li $v0, 4
        syscall
```

```
	la $a0, menuOp6
	li $v0, 4
	syscall

	la $a0, menuOp7
	li $v0, 4
	syscall

	la $a0, menuOp8
	li $v0, 4
	syscall

	la $a0, menuTitle
	li $v0, 4
	syscall

	li $v0, 5
	syscall

	beq $v0, 1, option1
	beq $v0, 2, option2
	beq $v0, 3, option3
	beq $v0, 4, option4
	beq $v0, 5, option5
	beq $v0, 6, option6
	beq $v0, 7, option7
	beq $v0, 8, option8
	j __main

# reading dimensions
option1:
	# Getting N
	la $a0, inputDim
	li $v0, 4
	syscall

	li $v0, 5
	syscall

	move $s0, $v0 # $s0 - N
	mul $t0, $s0, $s0 # $t0 - NxN

	# Allocating in heap
```

```
        move $a0, $t0
        li $v0, 9
        syscall

        move $s1, $v0 # $s1 - starting address of array in heap

        j __main

option2:
        la $a0, inputMatrix
        li $v0, 4
        syscall


        move $t0, $s0 # n
        move $t1, $s1 # starting address
        mul $t0, $t0, $t0

        addi $t2, $zero, 1 # i - row
        addi $t3, $zero, 1 # j - column

        addi $t4, $zero, 0
        addi $t5, $zero, 0

outerWhile: # i
        bgt $t2, $s0, outerWhileDone
        innerWhile: # j
                bgt $t3, $s0, innerWhileDone
                move $t4, $t3
                addi $t4, $t4, -1
                mul $t4, $t4, $s0
                sll $t4, $t4, 2
                move $t5, $t2
                addi $t5, $t5, -1
                sll $t5, $t5, 2
                add $t6, $t4, $t5

                add $t7, $t6, $t1
                li $v0, 5
                syscall
                sw $v0, 0($t7)

                addi $t3, $t3, 1
```

```
                j innerWhile
        innerWhileDone:
                addi $t2, $t2, 1
                addi $t3, $zero, 1
                j outerWhile
outerWhileDone:
        j __main

option3:
        la $a0, enterPosI
        li $v0, 4
        syscall

        li $v0, 5
        syscall
        move $t0, $v0 # t0 - i

        la $a0, enterPosJ
        li $v0, 4
        syscall

        li $v0, 5
        syscall
        move $t1, $v0 #t1 - j

        addi $t1, $t1, -1
        mul $t1, $t1, $s0
        sll $t1, $t1, 2
        addi $t0, $t0, -1
        sll $t0, $t0, 2
        add $a0, $t1, $t0

        add $a0, $a0, $s1

        lw $a0, 0($a0)
        li $v0, 1
        syscall

        la $a0, newLine
        li $v0, 4
        syscall

        j __main
```

```
option4:
        # row major
        move $t0, $s0 # n
        move $t1, $s1 # starting address

        addi $s2, $zero, 0

        addi $t2, $zero, 1 # i
        addi $t3, $zero, 1 # j

        addi $t4, $zero, 0
        addi $t5, $zero, 0

outerWhileCM: # i
        bgt $t2, $s0, outerWhileCMDone
        innerWhileCM: # j
                bgt $t3, $s0, innerWhileCMDone
                addi $t4, $t2, -1
                mul $t4, $t4, $s0
                sll $t4, $t4, 2

                addi $t5, $t3, -1
                sll $t5, $t5, 2
                add $a0, $t4, $t5
                add $a0, $s1, $a0
                lw $a0, 0($a0)
                add $s2, $s2, $a0
                addi $t3, $t3, 1
                j innerWhileCM
        innerWhileCMDone:
                addi $t3, $zero, 1
                addi $t2, $t2, 1
                j outerWhileCM
outerWhileCMDone:
        move $a0, $s2
        li $v0, 1
        syscall
        la $a0, newLine
        li $v0, 4
        syscall
        j __main
```

```
option5:
	addi $t2, $zero, 1 # i
	addi $t3, $zero, 1 # j

	addi $s3, $zero, 0

	addi $t4, $zero, 0
	addi $t5, $zero, 0

outerWhileRM: # i
	bgt $t2, $s0, outerWhileRMDone
	innerWhileRM: # j
		bgt $t3, $s0, innerWhileRMDone
		addi $t4, $t2, -1
		sll $t4, $t4, 2

		addi $t5, $t3, -1
		mul $t5, $t5, $s0
		sll $t5, $t5, 2

		add $a0, $t4, $t5

		add $a0, $s1, $a0
		lw $a0, 0($a0)
		add $s3, $s3, $a0

		addi $t3, $t3, 1
		j innerWhileRM
	innerWhileRMDone:
		addi $t3, $zero, 1
		addi $t2, $t2, 1
		j outerWhileRM
outerWhileRMDone:
	move $a0, $s3
	li $v0, 1
	syscall
	la $a0, newLine
	li $v0, 4
	syscall
	j __main

option6:
	la $a0, selectRowOrColumn
```

```
        li $v0, 4
        syscall

        li $v0, 5
        syscall

        move $t4, $v0 # t4 - row col no

        beq $v0, $zero, rowDisplay
        beq $v0, 1, columnDisplay

rowDisplay:
        la $a0, enterRowOrColumnNumber
        li $v0, 4
        syscall

        li $v0, 5
        syscall

        move $t2, $v0

        addi $t0, $zero, 1 # j
        move $t1, $s1 # starting address

        addi $t3, $zero, 0
        addi $t4, $zero, 0

loop:   bgt $t0, $s0, op6Done
        addi $t3, $t0, -1
        mul $t3, $t3, $s0
        sll $t3, $t3, 2
        addi $t4, $t2, -1
        sll $t4, $t4, 2
        add $t3, $t3, $t4
        add $t5, $t3, $s1

        lw $a0, 0($t5)
        li $v0, 1
        syscall
        la $a0, space,
        li $v0, 4
        syscall
        addi $t0, $t0, 1
```

```
        addi $t3, $zero, 0
        addi $t4, $zero, 0
        addi $t5, $zero, 0
        j loop

columnDisplay:
        la $a0, enterRowOrColumnNumber
        li $v0, 4
        syscall

        li $v0, 5
        syscall

        move $t2, $v0

        addi $t0, $zero, 1 # j
        move $t1, $s1 # starting address

        addi $t3, $zero, 0
        addi $t4, $zero, 0

loopColumn:
        bgt $t0, $s0, op6Done
        addi $t3, $t0, -1

        sll $t3, $t3, 2
        addi $t4, $t2, -1
        mul $t4, $t4, $s0
        sll $t4, $t4, 2
        add $t3, $t3, $t4
        add $t5, $t3, $s1

        lw $a0, 0($t5)
        li $v0, 1
        syscall

        la $a0, newLine
        li $v0, 4
        syscall

        addi $t0, $t0, 1
        addi $t3, $zero, 0
        addi $t4, $zero, 0
```

```
        addi $t5, $zero, 0
        j loopColumn

op6Done:
        la $a0, newLine
        li $v0, 4
        syscall
        j __main

option7:
        #la $a0, enterN
        #li $v0, 4
        #syscall

        #li $v0, 5
        #syscall

        #move $s0, $v0 # s0 - n
        addi $s0, $s0, 50
        mul $a0, $s0, $s0
        li $v0, 9
        syscall
        move $s1, $v0

        # s0 -n s1- starting addres
        move $t0, $s0 # n
        move $t1, $s1 # starting address
        mul $t0, $t0, $t0

        addi $t2, $zero, 1 # i - row
        addi $t3, $zero, 1 # j - column

        addi $t4, $zero, 0
        addi $t5, $zero, 0

outerWhileN: # i
        bgt $t2, $s0, outerWhileDoneN
        innerWhileN: # j
                bgt $t3, $s0, innerWhileDoneN
                move $t4, $t3
                addi $t4, $t4, -1
                mul $t4, $t4, $s0
                sll $t4, $t4, 2
```

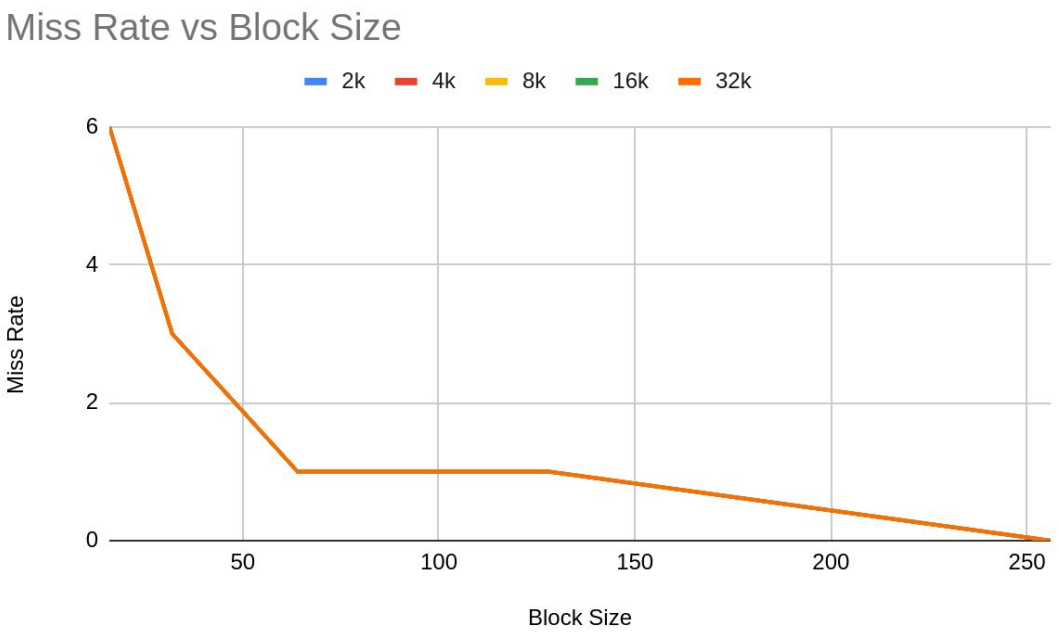
```
                move $t5, $t2
                addi $t5, $t5, -1
                sll $t5, $t5, 2
                add $t6, $t4, $t5
                add $t7, $t6, $t1
                li $a1, 10
                li $v0, 42
                syscall
                sw $a0, 0($t7)
                addi $t3, $t3, 1
                j innerWhileN
        innerWhileDoneN:
                addi $t2, $t2, 1
                addi $t3, $zero, 1
                j outerWhileN
outerWhileDoneN:
        j __main

option8:
        li $v0, 10
        syscall

        .data
        newLine: .asciiz "\n"
        menuTitle: .asciiz "What would you like to do? "
        menuOp1: .asciiz "1. Enter the dimensions of the matrix.\n"
        menuOp2: .asciiz "2. Enter the contents of the matrix(one at a time and row
wise). \n"
        menuOp3: .asciiz "3. Enter position to access the matrix element at this
position.\n"
        menuOp4: .asciiz "4. Get row major summation of matrix.\n"
        menuOp5: .asciiz "5. Get column major summation of matrix.\n"
        menuOp6: .asciiz "6. Get whole column or row.\n"
        menuOp7: .asciiz "7. Create a matrix of size N and initialize randomly.\n"
        menuOp8: .asciiz "8. Exit\n"
        inputDim: .asciiz "Dimension N for NxN matrix: "
        inputMatrix: .asciiz "Enter contents of matrix one-by-one row wise: \n"
        enterPosI: .asciiz "Enter the row number: "
        enterPosJ: .asciiz "Enter the column number: "
        enterN: .asciiz "Enter the value of N: "
        selectRowOrColumn: .asciiz "Enter 0 for row display, 1 for column display: "
        enterRowOrColumnNumber: .asciiz "Enter row or column number: "
        space: .asciiz " "
```

PART 2

a.

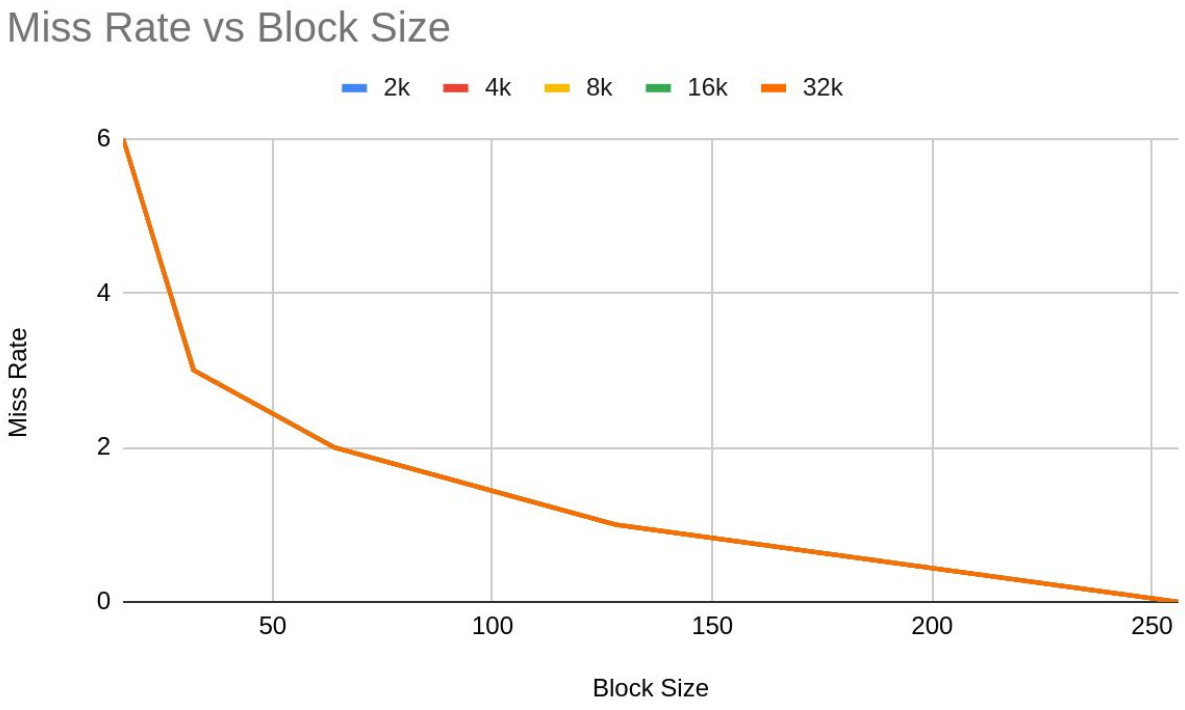
Row Major Addition
Matrix Size: 50 x 50

	Block Size (number of words)				
Cache Size (bytes)	**16**	**32**	**64**	**128**	**256**
2048	Miss Rate: 6% # of Misses: 163	Miss Rate: 3% # of Misses: 82	Miss Rate: 1% # of Misses: 42	Miss Rate: 1% # of Misses: 21	Miss Rate: 0% # of Misses: 11
4096	Miss Rate: 6% # of Misses: 163	Miss Rate: 3% # of Misses: 82	Miss Rate: 1% # of Misses: 42	Miss Rate: 1% # of Misses: 21	Miss Rate: 0% # of Misses: 11
8192	Miss Rate: 6% # of Misses: 163	Miss Rate: 3% # of Misses: 82	Miss Rate: 1% # of Misses: 42	Miss Rate: 1% # of Misses: 21	Miss Rate: 0% # of Misses: 11
16384	Miss Rate: 6% # of Misses: 163	Miss Rate: 1% # of Misses: 82	Miss Rate: 1% # of Misses: 42	Miss Rate: 1% # of Misses: 21	Miss Rate: 0% # of Misses: 11
32768	Miss Rate: 6% # of Misses: 163	Miss Rate: 3% # of Misses: 82	Miss Rate: 1% # of Misses: 42	Miss Rate: 1% # of Misses: 21	Miss Rate: 0% # of Misses: 11



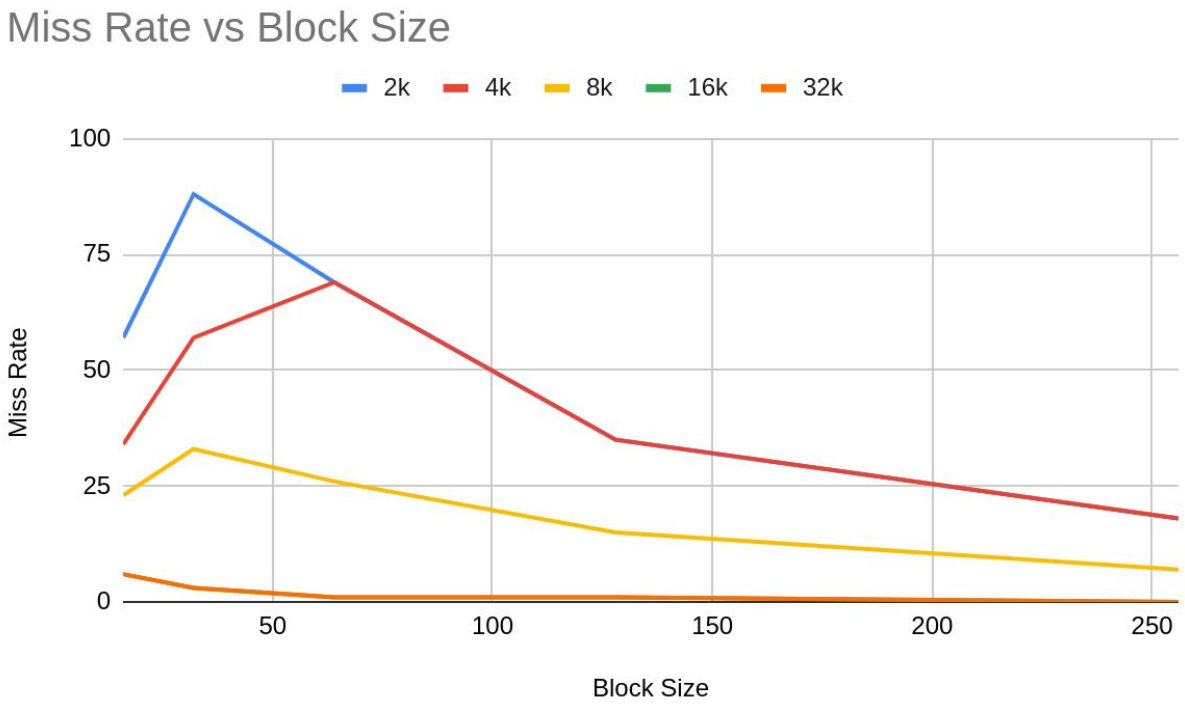
Matrix 100 x 100

	Block Size (number of words)				
Cache Size (bytes)	**16**	**32**	**64**	**128**	**256**
2048	Miss Rate: 6% # of Misses: 631	Miss Rate: 3% # of Misses: 316	Miss Rate: 2% # of Misses: 159	Miss Rate: 1% # of Misses: 80	Miss Rate: 0% # of Misses: 41
4096	Miss Rate: 6% # of Misses: 631	Miss Rate: 3% # of Misses: 316	Miss Rate: 2% # of Misses: 159	Miss Rate: 1% # of Misses: 80	Miss Rate: 0% # of Misses: 41
8192	Miss Rate: 6% # of Misses: 631	Miss Rate: 3% # of Misses: 316	Miss Rate: 2% # of Misses: 159	Miss Rate: 1% # of Misses: 80	Miss Rate: 0% # of Misses: 41
16384	Miss Rate: 6% # of Misses: 631	Miss Rate: 3% # of Misses: 316	Miss Rate: 2% # of Misses: 159	Miss Rate: 1% # of Misses: 80	Miss Rate: 0% # of Misses: 41
32768	Miss Rate: 6% # of Misses: 631	Miss Rate: 3% # of Misses: 316	Miss Rate: 2% # of Misses: 159	Miss Rate: 1% # of Misses: 80	Miss Rate: 0% # of Misses: 41



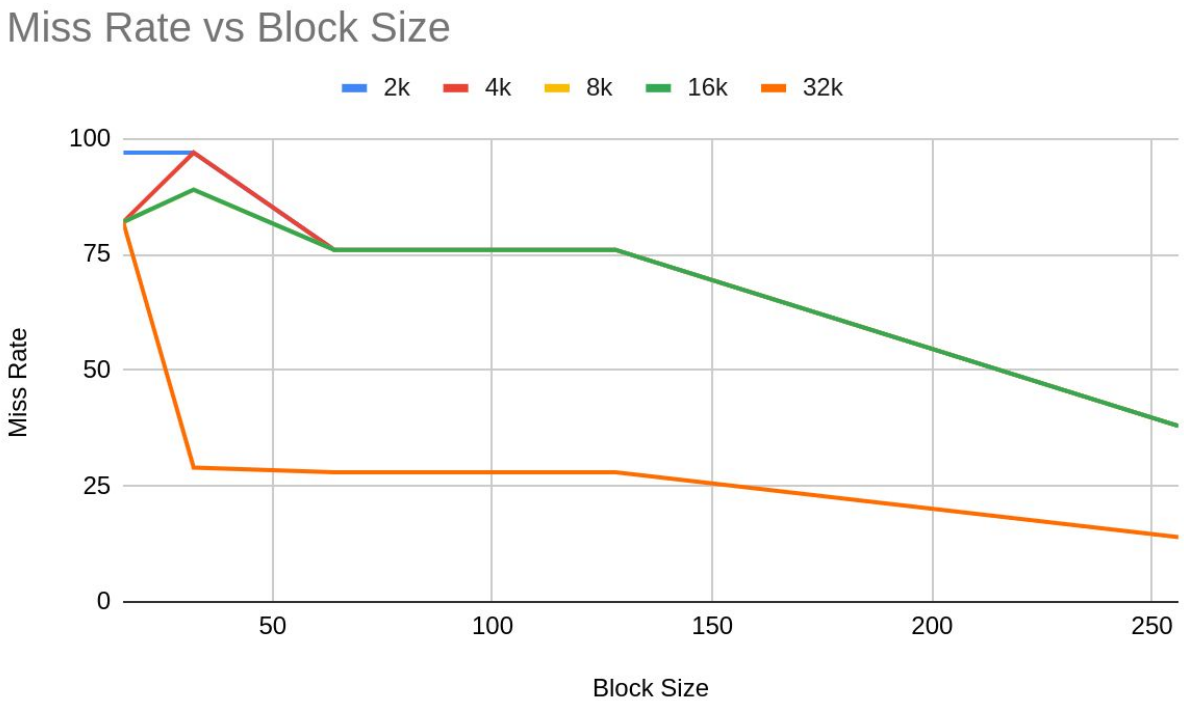
Column Major Addition
Matrix 50 x 50

	Block Size (number of words)				
Cache Size (bytes)	**16**	**32**	**64**	**128**	**256**
2048	Miss Rate: 57% # of Misses: 1619	Miss Rate: 88% # of Misses: 2503	Miss Rate: 69% # of Misses: 1956	Miss Rate: 35% # of Misses: 1001	Miss Rate: 18% # of Misses: 501
4096	Miss Rate: 34% # of Misses: 978	Miss Rate: 57% # of Misses: 1636	Miss Rate: 69% # of Misses: 1956	Miss Rate: 35% # of Misses: 1001	Miss Rate: 18% # of Misses: 501
8192	Miss Rate: 23% # of Misses: 951	Miss Rate: 33% # of Misses: 926	Miss Rate: 26% # of Misses: 734	Miss Rate: 15% # of Misses: 413	Miss Rate: 7% # of Misses: 207
16384	Miss Rate: 6% # of Misses: 163	Miss Rate: 3% # of Misses: 82	Miss Rate: 1% # of Misses: 42	Miss Rate: 1% # of Misses: 21	Miss Rate: 0% # of Misses: 11
32768	Miss Rate: 6% # of Misses: 163	Miss Rate: 3% # of Misses: 82	Miss Rate: 1% # of Misses: 42	Miss Rate: 1% # of Misses: 21	Miss Rate: 0% # of Misses: 11



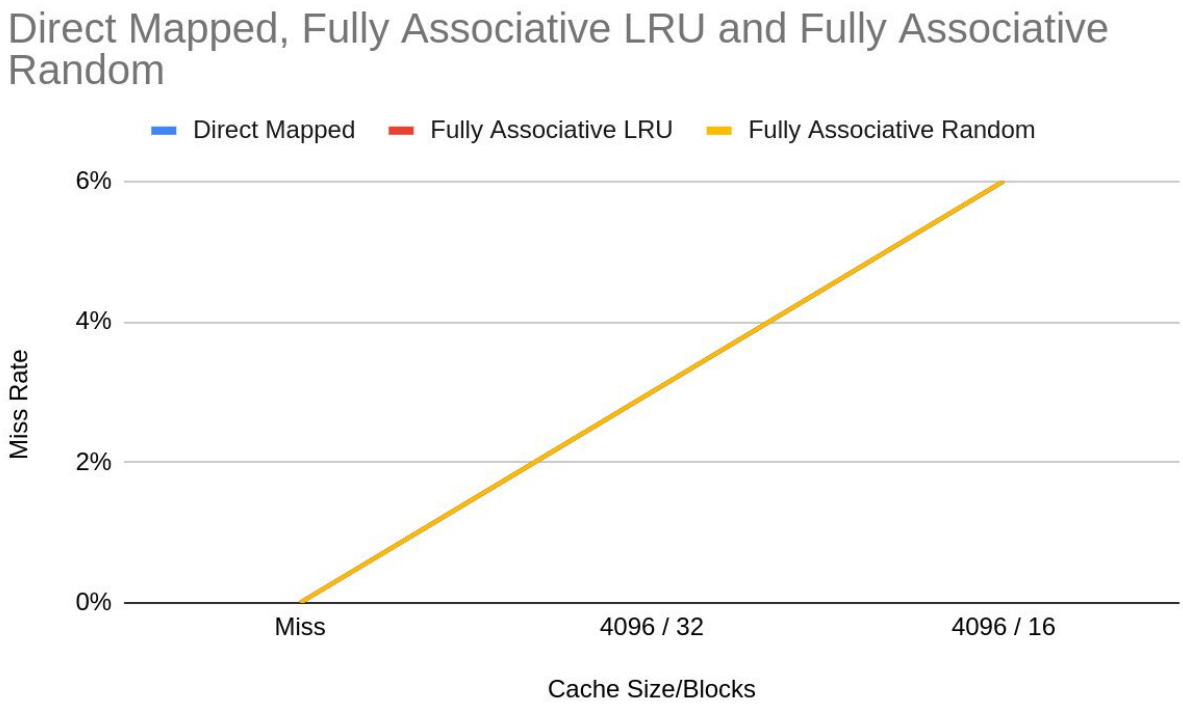
Matrix 100 x 100

	Block Size (number of words)				
Cache Size (bytes)	**16**	**32**	**64**	**128**	**256**
2048	Miss Rate: 97% # of Misses: 10006	Miss Rate: 97% # of Misses: 10003	Miss Rate: 76% # of Misses: 7817	Miss Rate: 76% # of Misses: 7817	Miss Rate: 38% # of Misses: 3917
4096	Miss Rate: 82% # of Misses: 8470	Miss Rate: 97% # of Misses: 10003	Miss Rate: 76% # of Misses: 7817	Miss Rate: 76% # of Misses: 7817	Miss Rate: 38% # of Misses: 3917
8192	Miss Rate: 82% # of Misses: 8470	Miss Rate: 89% # of Misses: 9235	Miss Rate: 76% # of Misses: 7817	Miss Rate: 76% # of Misses: 7817	Miss Rate: 38% # of Misses: 3917
16384	Miss Rate: 82% # of Misses: 8470	Miss Rate: 89% # of Misses: 9253	Miss Rate: 76% # of Misses: 7817	Miss Rate: 76% # of Misses: 7817	Miss Rate: 38% # of Misses: 3917
32768	Miss Rate: 82% # of Misses: 8470	Miss Rate: 29% # of Misses: 3024	Miss Rate: 28% # of Misses: 2882	Miss Rate: 28% # of Misses: 2882	Miss Rate: 14% # of Misses: 1457



b.

	Cache Type		
Cache Size/Block size	**Direct Mapped**	**Fully Associative LRU**	**Fully Associative Random**
4096 / 256 (Good)	0%	0%	0%
4096 / 32 (Medium)	3%	3%	3%
4096 / 16 (Poor)	6%	6%	6%



It can be seen from the graph above that the good, medium and poor hit rate values are exactly the same for direct mapped, fully associative LRU and fully associative random caches. A change from direct mapped cache to a fully associative cache using LRU does not cause a change in the miss rate. This is because when row major addition is performed, the neighbouring integers are taken from the memory and placed into cache due to spatial locality. The cache is large enough to store sufficient values from the array, therefore the miss rate is very low. A shift to fully associative random cash has similar reasoning. When an array element is accessed, due to spatial locality most of the neighbouring integers are also put into the cache, causing a low miss rate.

c.

Cache Size = 4096 bytes

Row Major Addition

Medium Hit Rate (Block size 32)

N-way Cache	2	4	8	16
Miss Rate	3%	3%	3%	3%
Hit Rate	97%	97%	97%	97%
Miss Count	82	82	82	82

Good Hit Rate (Block size 256)

N-way Cache	1	2	4	-
Miss Rate	0%	0%	0%	-
Hit Rate	100%	100%	100%	-
Miss Count	11	11	11	-

Note: For this configuration, 8 sets does not exist because the number of blocks in cache is 4.

Poor Hit Rate (Block size 16)

N-way Cache	2	4	8	16
Miss Rate	6%	6%	6%	6%
Hit Rate	94%	94%	94%	94%
Miss Count	163	163	163	163

Column Major Addition

Medium Hit Rate (Block size 32)

N-way Cache	2	4	8	16
Miss Rate	75%	88%	88%	88%
Hit Rate	25%	12%	12%	12%
Miss Count	2143	2503	2503	2503

Good Hit Rate (Block size 256)

N-way Cache	1	2	4	-
Miss Rate	18%	18%	18%	-
Hit Rate	82%	82%	82%	-
Miss Count	501	501	501	-

Note: For this configuration, 8 sets does not exist because the number of blocks in cache is 4.

Poor Hit Rate (Block size 16)

N-way Cache	2	4	8	16
Miss Rate	19%	16%	7%	7%
Hit Rate	81%	84%	93%	93%
Miss Count	539	451	206	206

For row major addition it can be seen that miss rate is the same for each of the three cases. This is because of spatial locality. When an item is accessed, its neighbouring items are also put into cache. Thus the hit rate is the same as we increase the number of sets.

For column major addition, as the value of N increases, it can be seen that to a certain point the miss rate decreases. After this point, it gets constant. This is because at this point, the cache is able to store all the neighbouring elements of the accessed item. Therefore, it contains most of the items. For the configuration in medium hit rate, N = 4 is the point where this occurs. For good and poor hit rate, 1 and 8 respectively seem to be this value. For the configuration of good hit rate, 1-way cache provides a good miss rate because it tends to store most of the items that are to be used in subsequent steps. And for poor hit-rate, N = 8 provide low miss rates.