CS224

SECTION 03

LAB06 PRELIMINARY REPORT

AYBÜKE CEREN DURAN/21302686

# Part1

## Question-01:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Cache**  **Size KB** | **N way**  **cache** | **Word**  **Size** | **Block size**  **(no. of words)** | **No. of**  **Sets** | **Tag Size**  **in bits** | **Index Size**  **(Set No.) in bits** | **Word Block**  **Offset**  **Size in bits1** | **Byte**  **Offset**  **Size in bits2** | **Block**  **Replacement**  **Policy Needed (Yes/No)** |
| 1 | 64 | 1 | 32 bits | 4 |  | 14 | 12 | 2 | 2 | NO |
| 2 | 64 | 2 | 32 bits | 4 |  | 17 | 11 | 2 | 2 | YES |
| 3 | 64 | 4 | 32 bits | 8 |  | 18 | 9 | 3 | 2 | YES |
| 4 | 64 | Full | 32 bits | 8 |  | 27 | 0 | 3 | 2 | YES |
| 9 | 128 | 1 | 16 bits | 4 |  | 15 | 14 | 2 | 1 | NO |
| 10 | 128 | 2 | 16 bits | 4 |  | 16 | 13 | 2 | 1 | YES |
| 11 | 128 | 4 | 16 bits | 16 |  | 18 | 10 | 4 | 0 | YES |
| 12 | 128 | Full | 16 bits | 16 |  | 28 | 0 | 4 | 0 | YES |

**1 Word Block Offset Size in bits:** Log2(No. of words in a block)

**2 Byte Offset Size in bits:** Log2(No. of bytes in a word)

## Question-02:

addi $t0, $0, 5

loop: beq $t0, $0, done

lw $t1, 0x4($0)

lw $t2, 0xC($0)

lw $t3, 0x8($0)

addi $t0, $t0, -1

j loop

done:

**a.** In the following table indicate the type of miss, if any: Compulsory, Conflict, Capacity.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Iteration No.** | | | | | |
| **1** | **2** | **3** | **4** | **5** |
| lw $t1, 0x4($0) | **Compulsory** | **Hit** | **Hit** | **Hit** | **Hit** |
| lw $t2, 0xC($0) | **Compulsory** | **Hit** | **Hit** | **Hit** | **Hit** |
| lw $t3, 0x8($0) | **Hit** | **Hit** | **Hit** | **Hit** | **Hit** |

**b.** What is the total cache memory size in number of bits? Include the V bit your calculations. Show the details of your calculation.

C=8 words

b=2 words

B=4

In direct mapping (N=1), S=B. Therefore, S=4 in our case.

Word block offset = log2(number of words within a block) = log22 =1

Number of set bits = log2(number of sets) = log24= 2

Byte offset = log2(number of bytes in a word) = log24=2

Tag = 32-(2+2+1) = 27bits

V = 1 bit Tag = 27 bits Data = 32 bits Data = 32 bits

Total cache contains = (1 + 27 +32 + 32) x 4 = 368 bits

**c.** State the number of AND and OR gates, EQUALITY COMPARATORs and MULTIPLEXERs needed to implement the cache memory.

2:1 Mux is needed for selecting the word within the block.

1 equality comparator is needed for checking tag matching.

1 AND gate is needed for checking the hit.

## Question-03:

Consider the above MIPS code segment. The cache capacity is 2 words, block size is 1 word. There is only 1 set. The block replacement policy is LRU.

**a.** In the following table indicate the type of miss, if any: Compulsory, Conflict, Capacity.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Iteration No.** | | | | | |
| **1** | **2** | **3** | **4** | **5** |
| lw $t1, 0x4($0) | **Compulsory** | **Capacity** | **Capacity** | **Capacity** | **Capacity** |
| lw $t2, 0xC($0) | **Compulsory** | **Capacity** | **Capacity** | **Capacity** | **Capacity** |
| lw $t3, 0x8($0) | **Capacity** | **Capacity** | **Capacity** | **Capacity** | **Capacity** |

**b.** How many bits are needed for the implementation of LRU policy? What is the total cache memory size in number of bits? Include the V bit and the bit(s) used for LRU in your calculations. Show the details of your calculation.

Word block offset = log2(number of words within a block) = log21 =0

Number of set bits = log2(number of sets) = 0

Byte offset = log2(number of bytes in a word) = log24=2

N=2

Tag = 32-2=30bits

Total cache contains = (1+30+32+1+30+32) x 1= 126 bits

**c.** State the number of AND and OR gates, EQUALITY COMPARATORs and MULTIPLEXERs needed to implement the cache memory.

2:1 Mux is needed for selecting the word within the block.

2 equality comparators are needed for checking tag matching.

2 AND and 1 OR gate are needed for checking the hit.

## Question-04:

TL1 =1 clock cycle

TL2= 5 clock cycles

TMM = 55 clock cycles

Miss L1 = 20%

Miss L2 = 5%

AMAT = TL1 + Miss L1 \* (TL2 +Miss L2 \*TMM) = 1 + 0.2 x ( 5+55x0.05) = 2.55 clock cycles

Clock rate with 4GHz = 0.25 ns

1012 instructions require time = 1012 x 0.25 x2.55 = 6.375 x 108

## Question-05:

.data

message: .asciiz "\nHello there!\n"

options: .asciiz "\nHere are your options: \n"

optionsDisplay: .asciiz " 1.To create default matrix(with dimension N) which has consecutive values in its columns\n 2. To create your matrix(with dimension N)\n 3.Enter the matrix element to be accessed and also be displayed:\n 4.Obtain sum of matrix elements by row-major (row by row) summation\n 5. Obtain the sum of matrix elements by column-major (column by column) summation\n 6.Display entire row or column\n 7.Quit\n"

userMatrixArray: .asciiz "Create an array with proper size\n"

enterN: .asciiz "Please enter N for allocation of default matrix\n"

userN: .asciiz "Please enter N for allocation of your matrix \n"

writeDone: .asciiz "Storing consecutive elements is successfully done!\n"

enterElement: .asciiz "Please enter your element into matrix: \n"

done: .asciiz "Storing user elements is successfully done!\n"

askRow: .asciiz "Please enter the row number which is i: \n"

askColumn: .asciiz "Please enter the column number: which is j: \n"

theElement: .asciiz "Here is the matrix[i][j]: \n"

printColByCol: .asciiz "Here is the summation of matrix elements column-major (column by column) summation: \n"

printRowByRow: .asciiz "Here is the summation of matrix elements row-major (row by row) summation: \n"

croc: .asciiz " Enter 1 for display a row... \nEnter 0 for display a column...\n"

enterRow: .asciiz "Please enter the row number to be displayed: \n"

enterColumn: .asciiz "Please enter the column number to be displayed: \n"

hereRow: .asciiz "Here is the row-"

hereColumn: .asciiz"Here is the column-"

ikinokta: .asciiz":\n"

newBosluk: .asciiz " "

.text

matrixMenu:

#we will assume that matrix elements will be placed column by column

li $v0,4#display message

la $a0,message

syscall

li $v0,4

la $a0,options

syscall

li $v0,4#display options

la $a0,optionsDisplay

syscall

li $v0,5

syscall

move $t7, $v0#copy user choice into $t7

beq $t7, 1, defaultMatrixAllocation

beq $t7, 2, userMatrixAllocation

beq $t7, 3, displayElement

beq $t7, 4, sumRowByRow

beq $t7, 5, sumColByCol

beq $t7, 6, displayRowOrColumn

beq $t7, 7, quit

defaultMatrixAllocation:

li $v0,4

la $a0, enterN

syscall

li $v0, 5

syscall #Now, read N

move $t8, $v0 #copy N for further use

move $t2, $v0 #copy N into $t2

mul $t3, $t2, $t2#In this line I calculated N^2

mul $a0, $t3, 4 #bytes to allocate matrix in heap

#memory allocation

li $v0,9

syscall

move $s2, $v0 #base address is in $s2 now...

move $t5, $t3#for further use, I copied N^2 into $t5

li $s5, 1#consecutive value = 1, in the function createDefaultMatrix it will be incremented

jal createDefaultMatrix

j promptMenu

createDefaultMatrix:

addi $sp, $sp, -12

sw $s2, 0($sp)#base address of the matrix is in the stack

sw $ra, 4($sp)#ra of the matrix is in the stack

sw $t5, 8($sp)#value of N^2 is in the stack

storeDefaultElements:

sgt $t4, $s5, $t5

beq $t4, 1, storeDone

sw $s5, 0($s2)

addi $s2, $s2, 4

addi $s5, $s5, 1

j storeDefaultElements

storeDone:

li $v0,4

la $a0, writeDone

syscall

lw $s2, 0($sp)#get original address of the base address of the matrix

lw $ra, 4($sp)# get $ra

lw $t5, 8($sp)#get tha value of N^2 from the stack

addi $sp, $sp, 12

jr $ra

userMatrixAllocation:

li $v0,4

la $a0, userN

syscall

li $v0, 5

syscall #Now, read N

move $t8, $v0 #copy N for further use

move $t2, $v0 #copy N into $t2

mul $t3, $t2, $t2#In this line I calculated N^2

mul $a0, $t3, 4 #bytes to allocate matrix in heap

#memory allocation

li $v0,9

syscall

move $s2, $v0 #base address is in $s2 now...

move $t5, $t3#for further use, I copied N^2 into $t5

move $s0, $t3#for further use, N^2 is copied into $s0

li $s5, 1#this will be used in iteration

jal createUserMatrix

j promptMenu

createUserMatrix:

addi $sp, $sp, -12

sw $s2, 0($sp)#base address of the matrix is in the stack

sw $ra, 4($sp)#ra of the matrix is in the stack

sw $t5, 8($sp)#value of N^2 is in the stack

storeUserElements:

sgt $t4, $s5, $t5

beq $t4, 1, storingCompleted

li $v0, 4

la $a0, enterElement

syscall

li $v0, 5#user will enter the element into matrix

syscall

move $t9, $v0

sw $t9, 0($s2)

addi $s2, $s2, 4

addi $s5, $s5, 1

j storeUserElements

storingCompleted:

li $v0,4

la $a0, done

syscall

lw $s2, 0($sp)#get original address of the base address of the matrix

lw $ra, 4($sp)# get $ra

lw $t5, 8($sp)#get tha value of N^2 from the stack

addi $sp, $sp, 12

jr $ra

displayElement:

#ask the row

li $v0, 4

la $a0, askRow

syscall

li $v0, 5

syscall

move $s6, $v0 #copy the row number into $s6 i

#ask the column

li $v0, 4

la $a0, askColumn

syscall

li $v0, 5

syscall

move $s7, $v0 #copy the column number into $s7 j

#now, calculate the position according to displacement formula

jal findDisplacement

move $s3, $v0

li $v0,4

la $a0, theElement

syscall

#display matrix[i][j]

lw $a0, 0($s3)

li $v0,1

syscall

j promptMenu

findDisplacement:

#formulation of the displacement is here: (j - 1) x N x 4 + (i - 1) x 4

subi $s7, $s7, 1 # j-1

mul $s7, $s7, $t2 # (j-1)\*N

mul $s7, $s7, 4#(j-1)\*N\*4

subi $s6, $s6, 1#(i-1)

mul $s6, $s6, 4#(i-1)\*4

add $t6, $s7, $s6

add $s3, $s2, $t6 #compute the address of the target element

move $v0, $s3 #put the address into return register

jr $ra

sumColByCol:

jal findSumColByCol

move $s7, $v0

li $v0,4

la $a0, printColByCol

syscall

add $a0, $s7, $zero

li $v0,1

syscall

j promptMenu

findSumColByCol:

li $s7, 0 #sum is initialized as 0

subi $sp, $sp, 16

sw $s2, 0($sp) # put the base address of the matrix into stack

sw $t5, 4($sp) #put the value of N^2 into the stack

sw $ra, 8($sp) #put the $ra into stack

sw $t8, 12($sp)

calculateSumColumn:

beqz $t5, calculationDone

lw $t1, 0($s2)

add $s7, $s7, $t1 #update the sum column by column

add $s2, $s2, 4

addi $t5, $t5, -1

j calculateSumColumn

calculationDone:

move $v0, $s7

lw $s2, 0($sp)

lw $t5, 4($sp)

lw $ra, 8($sp)

lw $t8, 12($sp)

addi $sp, $sp, 16

jr $ra

sumRowByRow:

jal findSumRowByRow

li $v0,4

la $a0, printRowByRow

syscall

add $a0, $s7, $zero

li $v0,1

syscall

j promptMenu

findSumRowByRow:

li $s7,0 #sum is initialized as 0

move $t4, $t8

move $a3, $s2

subi $sp, $sp, 16

sw $s2, 0($sp) # put the base address of the matrix into stack

sw $t5, 4($sp) #put the value of N^2 into the stack

sw $ra, 8($sp) #put the $ra into stack

sw $t8, 12($sp)

calculateSumRow:

li $a2, 1

move $t4, $t8

move $s2, $a3

beqz $t5, calculationDone2

internalLoop:

beqz $t4, calculateSumRowBefore

lw $t1, 0($s2)

add $s7, $s7, $t1 #update the value of the sum

mul $a2, $a2, $t8

mul $a2, $a2, 4

add $s2, $s2, $a2

subi $t4, $t4, 1

subi $t5, $t5, 1

li $a2, 1

j internalLoop

calculateSumRowBefore:

add $a3, $a3, 4

j calculateSumRow

calculationDone2:

lw $s2, 0($sp)

lw $t5, 4($sp)

lw $ra, 8($sp)

lw $t8, 12($sp)

addi $sp, $sp, 16

move $v0, $s7

jr $ra

promptMenu:

j matrixMenu

displayRowOrColumn:

li $v0, 4

la $a0, croc

syscall

li $v0, 5

syscall #enter

move $s7, $v0

beq $v0, 1, displayRow

beq $v0, 0, displayColumn

displayRow:

li $v0, 4

la $a0, enterRow

syscall

li $v0, 5

syscall

move $s7, $v0 #store row number in $s7

jal dRow

j matrixMenu

displayColumn:

li $v0, 4

la $a0, enterColumn

syscall

li $v0, 5

syscall

move $s7, $v0 #store column number in $s7

jal dColumn

j matrixMenu

dRow:

addi $sp, $sp, -12

sw $s2, 0($sp)#base address of the matrix is in the stack

sw $ra, 4($sp)

sw $t8, 8($sp)#value of N is in the stack

move $a2, $t8

li $v0, 4

la $a0, hereRow

syscall

li $v0,1

move $a0, $s7

syscall

li $v0, 4

la $a0, ikinokta

syscall

#calculate the base address of the row

calcAddresRow:

beq $s7, 1,forRow

add $s2, $s2, 4

subi $s7, $s7, 1

j calcAddresRow

forRow:

beq $a2, 0, dRowDone

lw $a0, 0($s2)#load row element

li $v0,1 #display row element

syscall

li $v0, 4

la $a0, newBosluk

syscall

after:

subi $a2, $a2, 1

mul $t4, $t8, 4

add $s2, $s2, $t4

j forRow

dRowDone:

lw $s2, 0($sp)

lw $ra, 4($sp)

lw $t8, 8($sp)

addi $sp, $sp, 12

jr $ra

dColumn:

addi $sp, $sp, -12

sw $s2, 0($sp)#base address of the matrix is in the stack

sw $ra, 4($sp)

sw $t8, 8($sp)#value of N is in the stack

li $v0, 4

la $a0, hereColumn

syscall

li $v0,1

move $a0, $s7

syscall

li $v0, 4

la $a0, ikinokta

syscall

#calculate the base address of the column

calcAddresCol:

beq $s7, 1,forColumn

mul $t4, $t8, 4

add $s2, $s2, $t4

subi $s7, $s7, 1

forColumn:

beq $t8, 0, dColumnDone

lw $a0, 0($s2)#load column element

li $v0,1 #display column element

syscall

li $v0, 4

la $a0, newBosluk

syscall

then:

subi $t8, $t8, 1

add $s2, $s2, 4

j forColumn

dColumnDone:

lw $s2, 0($sp)

lw $ra, 4($sp)

lw $t8, 8($sp)

addi $sp, $sp, 12

jr $ra

quit:

li $v0, 10

syscall

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Block Size(words)  Cache Size (Bytes) | 2 | 4 | 8 | 16 | 32 |
| 256(0.26KB) | Miss rate:  47%  Hit rate: 53%  # of misses: 10220 | Miss rate:  24%  Hit rate:  76%  # of misses:  5113 | Miss rate:  12%  Hit rate:  88%  # of misses:  2559 | Miss rate:  6%  Hit rate:  94%  # of misses:  1283 | Miss rate:  3%  Hit rate:  97%  # of misses:  647 |
| 512(0.5KB) | Miss rate: 47%  Hit rate:  53%  # of misses:  10218 | Miss rate:  24%  Hit rate:  76%  # of misses:  5112 | Miss rate:  12%  Hit rate:  88%  # of misses:  2559 | Miss rate:  6%  Hit rate:  94%  # of misses:  1283 | Miss rate:  3%  Hit rate:  97%  # of misses:  326 |
| 1024(1KB) | Miss rate: 47%  Hit rate:  53%  # of misses: 10218 | Miss rate:  24%  Hit rate:  76%  # of misses:  5112 | Miss rate:  12%  Hit rate:  88%  # of misses:  2558 | Miss rate:  6%  Hit rate:  94%  # of misses:  1282 | Miss rate:  3%  Hit rate:  97%  # of misses:  646 |
| 2048(2 KB) | Miss rate:  47%  Hit rate:  53%  # of misses:  10217 | Miss rate:  24%  Hit rate:  76%  # of misses:  5112 | Miss rate:  12%  Hit rate:  88%  # of misses:  2558 | Miss rate:  6%  Hit rate:  94%  # of misses:  1282 | Miss rate:  3%  Hit rate:  97%  # of misses:  646 |
| 4096(4 KB) | Miss rate:  47%  Hit rate:  53%  # of misses:  10218 | Miss rate:  24%  Hit rate:  53%  # of misses:  2239 | Miss rate:  12%  Hit rate:  88%  # of misses:  2558 | Miss rate:  6%  Hit rate:  94%  # of misses:  1282 | Miss rate:  3%  Hit rate: 97%  # of misses:  646 |

Table-01: Miss rate, hit rate and # of misses of column-wise summation for Direct Mapped Cache (N=100)

Graph-01: Direct Mapped Cache with N=100 for column-wise summation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Block Size(words)  Cache Size (Bytes) | 2 | 4 | 8 | 16 | 32 |
| 256(0.26KB) | Miss rate:  70%  Hit rate: 30%  # of misses: 15218 | Miss rate:  58%  Hit rate:  42%  # of misses:  12613 | Miss rate:  52%  Hit rate:  48%  # of misses:  11310 | Miss rate:  49%  Hit rate:  51%  # of misses:  10659 | Miss rate:  48%  Hit rate:  52%  # of misses:  10334 |
| 512(0.5KB) | Miss rate: 70%  Hit rate:  30%  # of misses:  15216 | Miss rate:  58%  Hit rate:  42%  # of misses:  12612 | Miss rate:  52%  Hit rate:  48%  # of misses:  11310 | Miss rate:  49%  Hit rate:  51%  # of misses:  10659 | Miss rate:  48%  Hit rate:  52%  # of misses:  10334 |
| 1024(1KB) | Miss rate: 64%  Hit rate:  36%  # of misses: 13816 | Miss rate:  49%  Hit rate:  51%  # of misses:  10052 | Miss rate:  48%  Hit rate:  52%  # of misses:  10333 | Miss rate:  49%  Hit rate:  51%  # of misses:  10658 | Miss rate:  48%  Hit rate:  52%  # of misses:  10333 |
| 2048(2 KB) | Miss rate:  31%  Hit rate:  69%  # of misses:  10718 | Miss rate:  24%  Hit rate:  76%  # of misses:  5099 | Miss rate:  43%  Hit rate:  57%  # of misses:  9204 | Miss rate:  50%  Hit rate:  50%  # of misses:  10655 | Miss rate:  33%  Hit rate:  67%  # of misses:  10650 |
| 4096(4 KB) | Miss rate:  31%  Hit rate:  69%  # of misses:  10716 | Miss rate:  24%  Hit rate:  76%  # of misses:  5099 | Miss rate:  29%  Hit rate:  71%  # of misses:  9489 | Miss rate:  42%  Hit rate:  58%  # of misses:  9117 | Miss rate:  47%  Hit rate:  53%  # of misses:  10331 |

Table-02: Miss rate, hit rate and # of misses of row-wise summation for Direct Mapped Cache (N=100)

Graph-02: Direct Mapped Cache with N=100 for row wise summation

PART B:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hit Rate(Good)  Cache Size(bytes):2048  Block Size(words):4 | Hit Rate(Medium)  Cache Size(bytes):2048  Block Size(words):16 | Hit Rate(Bad)  Cache Size(bytes):256  Block Size(words):2 |
| Fully Associative (LRU) | Cache hit rate: 52%  Miss rate: 48%  # of misses: 10332 | Cache hit rate: 50%  Miss rate: 50%  #of misses: 10655 | Cache hit rate: 29%  Miss rate: 71%  # of misses: 15199 |
| Fully Associative (Random) | Cache hit rate: 67%  Miss rate: 33%  # of misses: 7022 | Cache hit rate: 52%  Miss rate: 48%  #of misses: 10227 | Cache hit rate: 30%  Miss rate: 70%  #of misses: 14970 |
| Direct Mapped | Miss rate:24%  Hit rate:76%  # of misses:5099 | Miss rate:50%  Hit rate:50%  # of misses: 10655 | Miss rate:70%  Hit rate: 30%  # of misses: 15218 |

Table-03: Examines hit rate performances for various cache designs(N=100) for row major operation

Comparison of the results:

* The Direct Mapped good result versus the Fully Associative good result:

Hit rate increased, miss rate decreased when we apply Fully associative map (LRU).

When we apply random replacement, hit rate increases more compared to LRU.

* The Direct Mapped medium result versus the Fully Associative medium result:

Hit rate and miss rate did not changed when we apply Fully associative map(LRU).

When we apply random replacement, hit rate increased compared to LRU.

* The Direct Mapped poor result versus the Fully Associative poor result:

Hit rate decreased 1%, miss rate increased 1% when we apply Fully associative map(LRU).

When we apply random replacement, hit rate and miss rate did not changed.

Fully associative (random) made significant change on Direct Mapped (with good rate).

|  |  |  |  |
| --- | --- | --- | --- |
| N-way Set Associative  Cache Set Size | Hit Rate(Good)  Cache Size(bytes):2048  Block Size(words):4 | Hit Rate(Medium)  Cache Size(bytes):2048  Block Size(words):16 | Hit Rate(Bad)  Cache Size(bytes):256  Block Size(words):2 |
| 4 | Cache hit rate: 70%  Miss rate: 30%  # of misses: 6406 | Cache hit rate: 52%  Miss rate: 48%  # of misses: 10298 | Cache hit rate: 29%  Miss rate: 71%  # of misses: 15191 |
| 8 | Cache hit rate: 70%  Miss rate: 30%  # of misses: 6653 | Cache hit rate: 52%  Miss rate: 48%  # of misses: 10248 | Cache hit rate: 29%  Miss rate: 71%  # of misses: 15191 |
| 2 | Cache hit rate: 72%  Miss rate: 28%  # of misses: 5947 | Cache hit rate: 43%  Miss rate: 57%  # of misses: 12158 | Cache hit rate: 29%  Miss rate: 71%  # of misses: 15198 |
| 16 | Cache hit rate: 68%  Miss rate: 32%  # of misses: 6977 | Cache hit rate: 52%  Miss rate: 48%  # of misses: 10213 | Cache hit rate: 29%  Miss rate: 71%  # of misses: 15187 |

Table-04: Examines hit rate performances for n-way caches(N=100) for row major operation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Block Size(words)  Cache Size (Bytes) | 2 | 4 | 8 | 16 | 32 |
| 256(0.26KB) | Miss rate:  41%  Hit rate: 59%  # of misses: 2720 | Miss rate:  21%  Hit rate:  79%  # of misses:  2029 | Miss rate:  10%  Hit rate:  90%  # of misses:  685 | Miss rate:  5%  Hit rate:  95%  # of misses:  347 | Miss rate:  3%  Hit rate:  97%  # of misses:  179 |
| 512(0.5KB) | Miss rate: 41%  Hit rate:  59%  # of misses:  2718 | Miss rate:  20%  Hit rate:  80%  # of misses:  1362 | Miss rate:  10%  Hit rate:  90%  # of misses:  685 | Miss rate:  5%  Hit rate:  95%  # of misses:  347 | Miss rate:  3%  Hit rate:  97%  # of misses:  179 |
| 1024(1KB) | Miss rate: 41%  Hit rate:  59%  # of misses: 2718 | Miss rate:  20%  Hit rate:  80%  # of misses:  1362 | Miss rate:  10%  Hit rate:  90%  # of misses:  684 | Miss rate:  5%  Hit rate:  95%  # of misses:  346 | Miss rate:  3%  Hit rate:  97%  # of misses:  178 |
| 2048(2 KB) | Miss rate:  41%  Hit rate:  59%  # of misses:  2718 | Miss rate:  20%  Hit rate:  59%  # of misses:  1362 | Miss rate:  10%  Hit rate:  90%  # of misses:  684 | Miss rate:  5%  Hit rate:  95%  # of misses:  346 | Miss rate:  3%  Hit rate:  97%  # of misses:  178 |
| 4096(4 KB) | Miss rate:  41%  Hit rate:  59%  # of misses:  2718 | Miss rate:  20%  Hit rate:  59%  # of misses:  1362 | Miss rate:  10%  Hit rate:  90%  # of misses:  684 | Miss rate:  5%  Hit rate:  95%  # of misses:  346 | Miss rate:  3%  Hit rate: 97%  # of misses:  178 |

Table-05: Miss rate, hit rate and # of misses of column-wise summation for Direct Mapped Cache (N=50)

Graph-03: Direct Mapped Cache with N=50 for column-wise summation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Block Size(words)  Cache Size (Bytes) | 2 | 4 | 8 | 16 | 32 |
| 256(0.26KB) | Miss rate:  54%  Hit rate: 46%  # of misses: 3618 | Miss rate:  49%  Hit rate:  51%  # of misses:  3238 | Miss rate:  43%  Hit rate:  57%  # of misses:  2873 | Miss rate:  41%  Hit rate:  59%  # of misses:  2691 | Miss rate:  39%  Hit rate:  61%  # of misses:  2600 |
| 512(0.5KB) | Miss rate:  41%  Hit rate:  59%  # of misses:  2716 | Miss rate:  38%  Hit rate:  62%  # of misses:  2503 | Miss rate:  53%  Hit rate:  57%  # of misses:  2873 | Miss rate:  41%  Hit rate:  59%  # of misses:  2691 | Miss rate:  39%  Hit rate:  61%  # of misses:  2600 |
| 1024(1KB) | Miss rate:  41%  Hit rate:  59%  # of misses:  2711 | Miss rate:  28%  Hit rate:  72%  # of misses:  1828 | Miss rate:  29%  Hit rate:  71%  # of misses:  1896 | Miss rate:  40%  Hit rate:  60%  # of misses:  2690 | Miss rate:  39%  Hit rate:  61%  # of misses:  2599 |
| 2048(2 KB) | Miss rate:  41%  Hit rate:  59%  # of misses:  2711 | Miss rate:  27%  Hit rate:  73%  # of misses:  1814 | Miss rate:  21%  Hit rate:  79%  # of misses:  1380 | Miss rate:  27%  Hit rate:  73%  # of misses:  1803 | Miss rate:  39%  Hit rate:  61%  # of misses:  2599 |
| 4096(4 KB) | Miss rate:  41%  Hit rate:  59%  # of misses:  2711 | Miss rate:  27%  Hit rate:  73%  # of misses:  1814 | Miss rate:  20%  Hit rate:  80%  # of misses:  1339 | Miss rate:  17%  Hit rate:  83%  # of misses:  1141 | Miss rate:  26%  Hit rate: 74%  # of misses:  1726 |

Table-06: Miss rate, hit rate and # of misses of row-wise summation for Direct Mapped Cache (N=50)

Graph-04: Direct Mapped Cache with N=50 for row-wise summation

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hit Rate(Good)  Cache Size(bytes):4096  Block Size(words):16 | Hit Rate(Medium)  Cache Size(bytes):1024  Block Size(words):16 | Hit Rate(Bad)  Cache Size(bytes):256  Block Size(words):4 |
| Fully Associative (LRU) | Miss rate: 6%  Hit rate: 94%  # of misses: 386 | Miss rate: 41%  Hit rate: 59%  # of misses: 2687 | Miss rate: 50%  Hit rate: 50%  # of misses: 3229 |
| Fully Associative (Random) | Miss rate: 8%  Hit rate: 92%  # of misses: 1029 | Miss rate: 49%  Hit rate: 51%  # of misses: 6367 | Miss rate: 49%  Hit rate: 51%  # of misses: 6367 |
| Direct Mapped | Miss rate:17%  Hit rate:83%  # of misses:1141 | Miss rate:41%  Hit rate:59%  # of misses:5264 | Miss rate:49%  Hit rate:51%  # of misses:3238 |

Table-07: Examines hit rate performances for various cache designs(N=50) for row major operation

Comparison of the results:

* The Direct Mapped good result versus the Fully Associative good result:

Hit rate increased, miss rate decreased when we apply Fully associative map (LRU).

When we apply random replacement, hit rate increases more compared to LRU.

* The Direct Mapped medium result versus the Fully Associative medium result:

Hit rate and miss rate did not changed when we apply Fully associative map(LRU).

When we apply random replacement, hit rate increased compared to LRU.

* The Direct Mapped poor result versus the Fully Associative poor result:

Hit rate decreased 1%, miss rate increased 1% when we apply Fully associative map(LRU).

When we apply random replacement, hit rate and miss rate did not changed.

Fully associative (random) made significant change on Direct Mapped (with good rate).

|  |  |  |  |
| --- | --- | --- | --- |
| N-way Set Associative  Cache Set Size | Hit Rate(Good)  Cache Size(bytes):4096  Block Size(words):16 | Hit Rate(Medium)  Cache Size(bytes):1024  Block Size(words):16 | Hit Rate(Bad)  Cache Size(bytes):256  Block Size(words):4 |
| 4 | Miss rate: 11%  Hit rate: 89%  # of misses: 697 | Miss rate: 40%  Hit rate: 60%  # of misses: 2606 | Miss rate: 51%  Hit rate: 49%  # of misses: 3174 |
| 8 | Miss rate: 11%  Hit rate: 89%  # of misses: 683 | Miss rate: 40%  Hit rate: 60%  # of misses: 2580 | Miss rate: 29%  Hit rate: 71%  # of misses: 3154 |
| 16 | Miss rate: 10%  Hit rate: 90%  # of misses: 635 | Miss rate: 40%  Hit rate: 60%  # of misses: 2573 | Miss rate: 48%  Hit rate: 52%  # of misses: 3131 |
| 2 | Miss rate: 18%  Hit rate: 82%  # of misses: 1137 | Miss rate: 40%  Hit rate: 60%  # of misses: 2607 | Miss rate: 49%  Hit rate: 51%  # of misses: 3178 |

Table-08: Examines hit rate performances for various N-way caches(N=50) for row major operation

PART B:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hit Rate(Good)  Cache Size(bytes):  Block Size(words): | Hit Rate(Medium)  Cache Size(bytes):  Block Size(words): | Hit Rate(Bad)  Cache Size(bytes):  Block Size(words): |
| Fully Associative (LRU) |  |  |  |
| Fully Associative (Random) |  |  |  |
| Direct Mapped |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| N-way Set Associative  Cache Set Size | Hit Rate(Good)  Cache Size(bytes):  Block Size(words): | Hit Rate(Medium)  Cache Size(bytes):  Block Size(words): | Hit Rate(Bad)  Cache Size(bytes):  Block Size(words): |
| 4 |  |  |  |
| 8 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hit Rate(Good)  Cache Size(bytes):  Block Size(words): | Hit Rate(Medium)  Cache Size(bytes):  Block Size(words): | Hit Rate(Bad)  Cache Size(bytes):  Block Size(words): |
| Fully Associative (LRU) |  |  |  |
| Fully Associative (Random) |  |  |  |
| Direct Mapped |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| N-way Set Associative  Cache Set Size | Hit Rate(Good)  Cache Size(bytes):  Block Size(words): | Hit Rate(Medium)  Cache Size(bytes):  Block Size(words): | Hit Rate(Bad)  Cache Size(bytes):  Block Size(words): |
| 4 |  |  |  |
| 8 |  |  |  |