CS224
Section No.: 1
Fall 2019
Lab No.: 6
Berdan Akyürek/21600904

Preliminary Report

1.

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 bits**[1]	**Byte Offset Size in bits**[2]	**Block Replacement Policy Needed (Yes/No)**
1	64	1	32 bits	4	2^{12}	16	12	2	2	No
2	64	2	32 bits	4	2^{11}	17	11	2	2	Yes
3	64	4	32 bits	8	2^{9}	18	9	3	2	Yes
4	64	Full	32 bits	8	1	27	0	3	2	Yes
9	128	1	16 bits	4	2^{14}	15	14	2	1	No
10	128	2	16 bits	4	2^{13}	16	13	2	1	Yes
11	128	4	16 bits	16	2^{10}	17	10	4	1	Yes
12	128	Full	16 bits	16	1	27	0	4	1	Yes

2.

a.

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)	Conflict	Hit	Hit	Hit	Hit

b.

Tag size:
27 bit *4 = 108

cache memory: 368

c.

2 equality
1 mux
1 and
1 or

3.

a.

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.

There are 2 words* 32 bits = 64*32
Cache memory size: 2048 bits.

c.

1 or
2 equality
2 and
4x1 mux

4.

MR 1 = 0.2 and MR 2 = 0.05
Putting the values in AMAT = 2.2 clock cycles
4 * 10 9 / time = 10 12 = 4 * 10 -3

5.

#Berdan Akyurek
#21600904

# Registers and their contents
$s0: Beginning of the matrix array
$s1: menu choice
$s2: matrix size(N)
$s3: allocated memory size

# set values as 0

```
li $s0, 0
li $s1, 0
li $s2, 0
li $s3, 0
main:

            # Print menu
            la $a0, menu
        li $v0, 4
        syscall

            # Read and store menu choice
            li $v0, 5
        syscall
        move $s1, $v0

        beq $s1, 1, setMatSize
        beq $s1, 2, allocate
        beq $s1, 3, accessElement
        beq $s1, 4, rowMajor
        beq $s1, 5, colMajor
        beq $s1, 6, displayRowCol
        beq $s1, 7, exit

        # print invalid menu choice warning
        la $a0, invalidMenu
        li $v0, 4
        syscall

        j main


setMatSize:
            # ask for number
            la $a0, enterMatSize
        li $v0, 4
        syscall

        # read number and store
        li $v0, 5
        syscall
        move $s2, $v0

        bgt $s2, 0, sizeOk

        la $a0, invalidSize
        li $v0, 4
        syscall
        j setMatSize
        sizeOk:

        #Done messagge
```

```
    la $a0, done1
    li $v0, 4
    syscall

    move $a0, $s2
    li $v0, 1
    syscall

    la $a0, endl
    li $v0, 4
    syscall

      j main

allocate:

      bne $s2, 0,  allocOk

      la $a0, allocNo
    li $v0, 4
    syscall

      j main

      allocOk:

      # allocate new memory
      alloc:

      mul $s3, $s2, $s2
      mul $s3, $s3, 4
      move $a0, $s3
      li $v0, 9
      syscall

      move $s0, $v0

      la $a0, allocated
    li $v0, 4
    syscall

      # fill the matrix

      #int number = 1;
      #for(int i = 0 ; j < N ; i ++) //sutun
      #{
      #       for( int i = 0 ; i < N ; i ++ ) //satir
      #       {
      #               matrix[i][j] = number;
      #               number ++;
      #       }
      #}
```

```
	# displacement = (j - 1) x N x 4 + (i - 1) x 4

	li $t0, 1 # j = sutun
	li $t3, 1 # number to add
	loopFillMat1:
	bgt $t0, $s2, endLoopFillMat1

		li $t1, 1 # i = satir
		loopFillMat2:
		bgt $t1, $s2, endLoopFillMat2

			# matrix[j][i] = number;
			addi $t4, $t0, -1
			mul $t4, $t4, 4
			mul $t4, $t4, $s2

			addi $t5, $t1, -1
			mul $t5, $t5, 4
			add $t4, $t4, $t5

			# $t4 has the displacement now

			add $t4, $t4, $s0

			# $t4 has the address

			sw $t3, 0($t4)
		addi $t1, $t1, 1
		addi $t3, $t3, 1
		j loopFillMat2
		endLoopFillMat2:
	addi $t0, $t0, 1
	j loopFillMat1
	endLoopFillMat1:

	#display matrixFilled

	la $a0, matrixFilled
      li $v0, 4
      syscall

	j main

accessElement:

	bne $s3, 0, okk

	la $a0, notAllocYet
      li $v0, 4
      syscall
      j main
```

```
    okk:
    # ask for row and store in $t0
    la $a0, row
li $v0, 4
syscall

li $v0, 5
syscall
move $t0, $v0

blt $t0, $s2, accessROK
la $a0, invalidRow
li $v0, 4
syscall
j accessElement
accessROK:
bgt $t0, -1, askCol
la $a0, invalidRow
li $v0, 4
syscall
j accessElement
# ask for col and store in $t1
askCol:
    la $a0, col
li $v0, 4
syscall

li $v0, 5
syscall
move $t1, $v0

#####
blt $t1, $s2, accessCOK
la $a0, invalidRow
li $v0, 4
syscall
j askCol
accessCOK:
bgt $t1, -1, accessCOK2
la $a0, invalidCol
li $v0, 4
syscall
j askCol
accessCOK2:
#####

# displacement = (j - 1) x N x 4 + (i - 1) x 4
# $t0 = row(i)
# $t1 = col(j)

# calculate address
```

```
        addi $t2, $t1, -1
        mul $t2, $t2, 4
        mul $t2, $t2, $s2

        addi $t3, $t0, -1
        mul $t3, $t3, 4

        add $t2, $t2, $t3
        add $t2, $t2, $s0

        # load value
        lw $t4, 0($t2)

            # print result

            la $a0, theValin
        li $v0, 4
        syscall

        move $a0, $t0
        li $v0, 1
        syscall

        la $a0, comma
        li $v0, 4
        syscall

        move $a0, $t1
        li $v0, 1
        syscall

        la $a0, clBrac
        li $v0, 4
        syscall

        la $a0, is
        li $v0, 4
        syscall

        move $a0, $t4
        li $v0, 1
        syscall

        la $a0, endl
        li $v0, 4
        syscall

            j main

rowMajor:

            bne $s3, 0, okkRm
```

```
    la $a0, notAllocYet
li $v0, 4
syscall
j main

    okkRm:
    # ask for row and store in $t0

li $t0, 1

rmOK1:

bgt $t0, -1, rmOK2
la $a0, invalidRow
li $v0, 4
syscall
j rowMajor

rmOK2:


# $t8: all sum
# $t0: row
# $t1: loop counter (col)
# $t7: sum
li $t8, 0
rmLoop2:
bgt $t0, $s2, endRmLoop2
li $t1, 1
li $t2, 0
li $t7, 0

rmLoop:
bgt $t1, $s2, endRmLoop
    # calculate address

    addi $t2, $t1, -1
    mul $t2, $t2, 4
    mul $t2, $t2, $s2

    addi $t3, $t0, -1
    mul $t3, $t3, 4

    add $t2, $t2, $t3
    add $t2, $t2, $s0

    # load value
    lw $t4, 0($t2)

    # add to sum
    add $t7, $t7, $t4
```

```
        addi $t1, $t1, 1
        j rmLoop
        endRmLoop:

        # print sum
        la $a0, sumOfRow
        li $v0, 4
        syscall

        move $a0, $t0
        li $v0, 1
        syscall

        la $a0, is
        li $v0, 4
        syscall

        move $a0, $t7
        li $v0, 1
        syscall

        la $a0, endl
        li $v0, 4
        syscall

        add $t8, $t8, $t7
        addi $t0, $t0, 1

        j rmLoop2
        endRmLoop2:

        la $a0, sumAll
        li $v0, 4
        syscall

        move $a0, $t8
        li $v0, 1
        syscall

        la $a0, endl
        li $v0, 4
        syscall

           j main

colMajor:

           bne $s3, 0, okkCm

           la $a0, notAllocYet
        li $v0, 4
        syscall
        j main

           okkCm:

           li $t1, 0
           li $t8, 0
           mul $t2, $s2, $s2
           move $t3, $s0
           DDD:
           beq $t8, $t2, endDDD
           lw $t4, 0($t3)

           add $t1, $t1, $t4
           addi $t3, $t3, 4
           addi $t8, $t8, 1
           j DDD
           endDDD:

        # print all sum

        la $a0, sumAll2
        li $v0, 4
        syscall

        move $a0, $t1
        li $v0, 1
        syscall

        la $a0, endl
        li $v0, 4
        syscall


           j main

displayRowCol:

           bne $s3, 0, okkDRC

           la $a0, notAllocYet
        li $v0, 4
        syscall
        j main

           okkDRC:

           la $a0, rOrC
        li $v0, 4
        syscall

           li $v0, 5
        syscall
```

```
move $t9, $v0

beq $t9, 1, dispRow
beq $t9, 2, dispCol

la $a0, invalidMenu
li $v0, 4
syscall

j displayRowCol

dispRow:
    # ask for row and store in $t0
                la $a0, row
    li $v0, 4
    syscall

    li $v0, 5
    syscall
    move $t0, $v0

    blt $t0, $s2, drOK1
    la $a0, invalidRow
    li $v0, 4
    syscall
    j dispRow

    drOK1:

    bgt $t0, -1, drOK2
    la $a0, invalidRow
    li $v0, 4
    syscall
    j dispRow

    drOK2:

    la $a0, rowN
    li $v0, 4
    syscall

    move $a0, $t0
    li $v0, 1
    syscall

    la $a0, has
    li $v0, 4
    syscall

    # $t0: row
    # $t1: loop counter (col)
    # $t7: value
```

```
   li $t1, 0
            li $t2, 0
            li $t7, 0

            drLoop:
            beq $t1, $s2, enddrLoop
                    # calculate address

            addi $t2, $t1, -1
            mul $t2, $t2, 4
                    mul $t2, $t2, $s2

            addi $t3, $t0, -1
            mul $t3, $t3, 4

            add $t2, $t2, $t3
            add $t2, $t2, $s0

            # load value
            lw $t4, 0($t2)

            # print
            move $a0, $t4
            li $v0, 1
            syscall

            la $a0, space
            li $v0, 4
            syscall

            addi $t1, $t1, 1
   j drLoop
   enddrLoop:

   la $a0, endl
   li $v0, 4
   syscall

   j main
dispCol:
            #FILL

            # ask for col and store in $t1
            la $a0, col
   li $v0, 4
   syscall

   li $v0, 5
   syscall
   move $t1, $v0
```

```
blt $t1, $s2, dcOK1
la $a0, invalidRow
li $v0, 4
syscall
j dispCol

dcOK1:

bgt $t1, -1, dcOK2
la $a0, invalidRow
li $v0, 4
syscall
j dispCol

dcOK2:

la $a0, colN
li $v0, 4
syscall

move $a0, $t1
li $v0, 1
syscall

la $a0, has
li $v0, 4
syscall

        # $t0: loop counter (row)
# $t1: col
# $t7: value

li $t0, 0
        li $t2, 0
        li $t7, 0

        dcLoop:
        beq $t0, $s2, enddcLoop
                # calculate address

        addi $t2, $t1, -1
        mul $t2, $t2, 4
                mul $t2, $t2, $s2

        addi $t3, $t0, -1
        mul $t3, $t3, 4

        add $t2, $t2, $t3
        add $t2, $t2, $s0

        # load value
        lw $t4, 0($t2)
```

```
                # print
                move $a0, $t4
                li $v0, 1
                syscall

                la $a0, space
                li $v0, 4
                syscall

                addi $t0, $t0, 1
        j dcLoop
        enddcLoop:

        la $a0, endl
        li $v0, 4
        syscall

                j main

exit:

        # print goodbye message

        la $a0, goodbye
      li $v0, 4
      syscall

        # end program
        li $v0, 10
        syscall

.data
        endl: .asciiz "\n"
        menu: .asciiz "------------------ \n1. Enter matrix size \n2. Allocate an array \n3. Acces a matrix element \n4. Row major summation \n5. Column major summation \n6. Display a whole row or column \n7. Exit \n------------------ \nEnter your choice: "
        goodbye: .asciiz "Program is finished. Goodbye...\n"
        invalidMenu: .asciiz "You entered an invalid menu option! \n"
        enterMatSize: .asciiz "Enter the new matrix size: "
        done1: .asciiz "Matrix size successfully setted as "
        invalidSize: .asciiz "Matrix size cannot be less than 1! \n"
        allocNo: .asciiz "Matrix size did not specified yet! \n"
        deallocated: .asciiz "Old matrix is deallocated! \n"
        allocated: .asciiz "A new matrix is allocated! \n"
        matrixFilled: .asciiz "New matrix is filled with values! \n"
        row: .asciiz "Row: "
        col: .asciiz "Col: "
        theValin: .asciiz "The value in ("
        is: .asciiz " is: "
        comma: .asciiz ","
        clBrac: .asciiz ")"
```

```
invalidRow: .asciiz "Warning! Invalid row! \n"
invalidCol: .asciiz "Warning! Invalid column! \n"
notAllocYet: .asciiz "The matrix is not allocated yet! \n"
sumOfRow: .asciiz "The sum of row number "
sumOfCol: .asciiz "The sum of column number "
rowN: .asciiz "Row number "
colN: .asciiz "Col number "
has: .asciiz " has: "
space: .asciiz " "
rOrC: .asciiz "Row or column? \n1. Row\n2. Column \nChoice: "
sumAll: .asciiz "Sum of all rows is: "
sumAll2: .asciiz "Sum of all columns is: "
```