Bilkent University
CS Department

CS 224 - Digital Design and Computer Architecture

Preliminary Design Report Lab 06

Section 04

Okan Şen 21202377

18/12/2019

1) .

No	Cache Size KB	N way Cache	Word Size	Block Size	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	64	1	32 bits	4	4096	16 bits	12 bits	2 bits	2 bits	No
2	64	2	32 bits	4	2048	17 bits	11 bits	2 bits	2 bits	Yes
3	64	4	32 bits	8	512	18 bits	9 bits	3 bits	2 bits	Yes
4	64	Full	32 bits	8	256	19 bits	8 bits	3 bits	2 bits	Yes
9	128	1	16 bits	4	8192	16 bits	13 bits	2 bits	1 bit	No
10	128	2	16 bits	4	4096	17 bits	12 bits	2 bits	1 bit	Yes
11	128	4	16 bits	16	512	18 bits	9 bits	4 bits	1 bit	Yes
12	128	Full	16 bits	16	64	21 bits	6 bits	4 bits	1 bit	Yes

2)

Instr	Iteration No	Iteration No	Iteration No	Iteration No	Iteration No
Instr	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)
block bit = 1
# of set = # of block/N => 4/1 = 4

Set bit = 2

00…..00 | 00 | 1 | 00
Tag set blck byte

In each set;
1 status bit + 27 tag bit + 64 bit data = 92 bits

There are 4 sets, therefore; 92 x 4 = 364 bits total cache memory

c) 1 Mux2 for block selection
1 equality comparator
1 and

3)

Instr	Iteration No	Iteration No	Iteration No	Iteration No	Iteration No
Instr	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) N = c/b => 2/1 = 2
V= 2 bit Byte = 2 bit Block = 0 bit set = 0 bit 32-2(byte)= 30 tag bit
LRU = 2 bit

In each set(which there is only 1 set in this case);
30*2 tag bit + 2 status bit + 64 bit data + 2 LRU bit = 128 bits total cache memory

c) 2 equality comparator and 2 and gates will be enough for this system

4)
L1: 1 clock cycle miss rate = 20%
L2: 5 clock cycle miss rate = 5%

Memory: 50 clock cycle

Amat = h1t1 + (1-h1)h2t2 + (1-h1)(1-h2)(m)
= (0.8)(1) + (0.2)(0.95)(5) + (0.2)(0.05)(50)
= 0.8 + 0.95 + 0.5
= 2.25 clock cycle

4 GHz Time Period = ¼ * 10^-9 = 0.25ns
AMAT = 0.25 * 225
= 0.56 ns

Time taken for 1 instruction = 0.56ns
Time taken for 10^12 = 0.56 * 10^-9 * 10^12
= 0.56 * 10^3
= 560 sec

5) .

```
##OKAN SEN


#

#
###################################################################
	.text


	.globl _matrix

_matrix:			# execution starts here

	li $s0, 0		# initialize pointer storage register to 0 (=Null pointer)

	la $a0, msg110		# put msg110 address into a0
	li $v0,4# system call to print
	syscall			#   out the msg110 string



##

##	Output the menu to the terminal,
```

```
##		and get the user's choice

##

##
MenuZ:
	la $a0, msg111		# put msg111 address into a0
	li $v0,4# system call to print
	syscall		#   out the msg111 string

	la $a0,msg112		# put msg112 address into a0
	li $v0,4# system call to print
	syscall		#   out the msg112 string


	la $a0, msg113		# put msg113 address into a0
	li $v0,4# system call to print
	syscall		#   out the msg113 string

	la $a0, msg114		# put msg114 address into a0
	li $v0,4# system call to print
	syscall		#   out the msg114 string


	la $a0, msg115		# put msg115 address into a0
	li $v0,4# system call to print
	syscall		#   out the msg115 string

	la $a0, msg116		# put msg116 address into a0
	li $v0,4# system call to print
	syscall		#   out the msg116 string

	la $a0, msg116.2	# put msg116 address into a0
	li $v0,4# system call to print
	syscall		#   out the msg116 string

	la $a0, msg118		# put msg118 address into a0
	li $v0,4# system call to print
	syscall		#   out the msg118 string

EnterChoice:
```

```
        la $a0, msg119          # put msg119 address into a0
        li $v0,4# system call to print
        syscall            #   out the msg119 string

        li $v0,5# system call to read
        syscall            # in the integer


        move $t9, $v0 # move choice into $t9





##

##

##      T1 through T7no use an if-else tree to test the user choice (in $s1)

##        and act on it by calling the correct routine

##

##

T1:     bne $t9, 1, T2 # if s1 = 1, do these things. Else go to T2 test
        ## Create N*N Size 2D Array

        la $a0, msg1.1          # ask for array size
        li $v0, 4
        syscall

        li $v0, 5                           # syscall 5 reads an integer
        syscall

        move $s1, $v0           # move input size to s1
        mul $a0, $s1, $s1       # store s1*s1 so that we can allocate a 2d array size using
a0
        li $v0, 9                           # allocate size in heap
        syscall

        move $s0, $v0           # put array address in s0
```

```
        move $s2, $a0               # also keep the s1*s1 in s2 for populating the array

        move $a0, $s0               # store array address in a0 for method call
        move $a1, $s1               # store dimension size in a1
        move $a2, $s2               # store N^2 in a2 for counter
        jal popMatrix



        j MenuZ

T2:     bne $t9, 2, T3 # if s1 = 2, do these things. Else go to T3 test
        ## Ask user the matrix element to be accessed and display the content
        move $a0, $s0
        move $a1, $s1
        jal printMatrix

        j MenuZ

T3:     bne $t9, 3, T4 # if s1 = 3, do these things. Else go to T4 test
        ## Obtain summation of matrix elements row-major (row by row) summation

        move $a0, $s0
        move $a1, $s1
        move $a2, $s2
        jal RMajorSum

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

        #print new line
        la $a0, endl
        li $v0, 4
        syscall

        #print new line
        la $a0, endl
        li $v0, 4
        syscall

        j MenuZ
```

```
T4:     bne $t9, 4, T5 # if s1 = 4, do these things. Else go to T5 test
        ## Obtain summation of matrix elements column-major (column by column)
summation
        move $a0, $s0
        move $a1, $s1
        move $a2, $s2
        jal CMajorSum

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

        #print new line
        la $a0, endl
        li $v0, 4
        syscall

        #print new line
        la $a0, endl
        li $v0, 4
        syscall

        j MenuZ


T5:     bne $t9, 5, T6 # if s1 = 5, do these things. Else go to T6 test
        ## Display desired elements of the matrix by specifying its row and column
member
        #ask for i
        la $a0, msg5.1
        li $v0, 4
        syscall

        # syscall 5 reads an integer
        li $v0, 5
        syscall

        move $t2, $v0

        #ask for j
        la $a0, msg5.2
```

```
        li $v0, 4
        syscall

        # syscall 5 reads an integer
        li $v0, 5
        syscall

        move $t3, $v0

        move $a0, $s0
        jal getByIndex

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

        #print new line
        la $a0, endl
        li $v0, 4
        syscall

        #print new line
        la $a0, endl
        li $v0, 4
        syscall

        j MenuZ

T6:     bne $t9, 6, T7 # if s1 = 6, do these things. Else go to T7 test

        la $a0, msg6.1
        li $v0, 4
        syscall

        la $a0, msg6.2
        li $v0, 4
        syscall

        # syscall 5 reads an integer
        li $v0, 5
        syscall
```

```
        move $t5, $v0

        la $a0, msg6.3
        li $v0, 4
        syscall

        # syscall 5 reads an integer
        li $v0, 5
        syscall

        move $t9, $v0

        move $a0, $s0
        move $a1, $s1
        move $a2, $s2
        jal printRC

        j MenuZ

T7:     bne $t9, 7, T7no
        ## exit this program
        li $v0, 10
        syscall

T7no:

        la $a0, msg128        # put msg128 address into a0
        li $v0, 4         # system call to print
        syscall           #   out the msg128 string

        j EnterChoice  # go to the place to enter the choice



#####################################################################

##



####################################################################
popMatrix:
```

```
	li $t1, 1		# counter for numbers
	move $t2, $a0		# start address displacement at 0

	lpop:
		bgt $t1, $a2, popDone
			sw $t1, ($t2)		# store t1 value in array's t2 displacement
			addi $t2, $t2, 4	# add 4 to get to next address of array
			addi $t1, $t1, 1	# add 1 to get to next value

			j lpop

popDone:
	li $t1, 0
	li $t2, 0
	jr $ra

# -----------------------------------------------------------------------------------------------------------
# -----------------------------------------------------------------------------------------------------------
printMatrix:
	li $t2, 1		# i = 1
	li $t3, 1		# j = 1
	li $v0, 0

	LPM:
		bgt $t3, $a1, printRowDone	# when we reach the end of the row reset it
and increase column
		bgt $t2, $a1, LPMDone		# if  we have reached the last row, finish
loop

		# the algorithm to get the offset from the beginning of the array address
		# returns the result in t5
		subi $t5, $t3, 1	# (j-1)
		mul $t5, $t5, $a1	# (j-1)xN
		mul $t5, $t5, 4		# (j-1)xNx4

		subi $t6, $t2, 1	# (i-1)
		mul $t6, $t6, 4		# (i-1)x4

		add $t5, $t5, $t6	# (j-1)xNx4 + (i-1)x4
		add $t6, $a0, $t5	# t6 = array address + offset

		lw $t1, ($t6)		#load the value we want to t1
		move $v0, $t1
```

```
		addi $t3, $t3, 1		# j++

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

		# print tab
		la $a0, tab
		li $v0, 4
		syscall

		move $a0, $s0			# reset a0 to array address

		j LPM

	printRowDone:
		addi $t2, $t2, 1		# i++
		li $t3, 1			# j = 1

		# print newline
		la $a0, endl
		li $v0, 4
		syscall

		move $a0, $s0			# reset a0 to array address

		j LPM

	LPMDone:
		li $t1, 0
		li $t2, 0
		li $t3, 0
		li $t5, 0
		li $t6, 0
		jr $ra
# ----------------------------------------------------------------------------------------------------------
# ----------------------------------------------------------------------------------------------------------
RMajorSum:
	li $t2, 1		# i = 1
	li $t3, 1		# j = 1
	li $v0, 0
```

```
	LRMS:
		bgt $t3, $a1, RowDone		# when we reach the end of the row reset it
and increase column
		bgt $t2, $a1, LRMSDone		# if  we have reached the last row, finish
loop

		# the algorithm to get the offset from the beginning of the array address
		# returns the result in t5
		subi $t5, $t3, 1		# (j-1)
		mul $t5, $t5, $a1		# (j-1)xN
		mul $t5, $t5, 4			# (j-1)xNx4

		subi $t6, $t2, 1		# (i-1)
		mul $t6, $t6, 4			# (i-1)x4

		add $t5, $t5, $t6		# (j-1)xNx4 + (i-1)x4
		add $t6, $a0, $t5		# t6 = array address + offset

		lw $t1, ($t6)			#load the value we want to t1
		add $v0, $v0, $t1		# keep the sum in v0

		addi $t3, $t3, 1		# j++
		j LRMS

	RowDone:
		addi $t2, $t2, 1		# i++
		li $t3, 1			# j = 1
		j LRMS

	LRMSDone:
		li $t1, 0
		li $t2, 0
		li $t3, 0
		li $t5, 0
		li $t6, 0
		jr $ra
# -------------------------------------------------------------------------------------------------------------
# -------------------------------------------------------------------------------------------------------------
CMajorSum:
	li $t2, 1		# i = 1
	li $t3, 1		# j = 1
	li $v0, 0
```

```
    LCMS:
        bgt $t2, $a1, ColumnDone     # when we reach the end of the column
reset it and increase row
        bgt $t3, $a1, LCMSDone       # if  we have reached the last column, finish
loop

        # the algorithm to get the offset from the beginning of the array address
        # returns the result in t5
        subi $t5, $t3, 1        # (j-1)
        mul $t5, $t5, $a1       # (j-1)xN
        mul $t5, $t5, 4         # (j-1)xNx4

        subi $t6, $t2, 1        # (i-1)
        mul $t6, $t6, 4         # (i-1)x4

        add $t5, $t5, $t6       # (j-1)xNx4 + (i-1)x4
        add $t6, $a0, $t5       # t6 = array address + offset

        lw $t1, ($t6)           #load the value we want to t1
        add $v0, $v0, $t1       # keep the sum in v0

        addi $t3, $t3, 1        # j++
        j LRMS

    ColumnDone:
        addi $t3, $t3, 1        # i++
        li $t2, 1               # j = 1
        j LRMS

    LCMSDone:
        li $t1, 0
        li $t2, 0
        li $t3, 0
        li $t5, 0
        li $t6, 0
        jr $ra
# ------------------------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------------------------
getByIndex:
    # the algorithm to get the offset from the beginning of the array address
    # returns the result in t5
    subi $t5, $t3, 1        # (j-1)
    mul $t5, $t5, $a1       # (j-1)xN
```

```
        mul $t5, $t5, 4         # (j-1)xNx4

        subi $t6, $t2, 1        # (i-1)
        mul $t6, $t6, 4         # (i-1)x4

        add $t5, $t5, $t6       # (j-1)xNx4 + (i-1)x4
        add $t6, $a0, $t5       # t6 = array address + offset

        lw $v0, ($t6)           #load the value we want to v0

        li $t2, 0
        li $t3, 0
        li $t5, 0
        li $t6, 0

        jr $ra
# ------------------------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------------------------
printRC:
        bne $t5, 1, pRCcheck
                move $t2, $t9
                li $t3, 1       # j = 1
                j pRCcont
        pRCcheck:
                move $t3, $t9
                li $t2, 1       # i = 1
                pRCcont:

                bne $t5, 1, pC          # if chosen decision is not row, then it is column
                LPRCr:
                        bgt $t3, $a1, LPRCDone
                        # returns the result in t5
                        subi $t5, $t3, 1        # (j-1)
                        mul $t5, $t5, $a1       # (j-1)xN
                        mul $t5, $t5, 4         # (j-1)xNx4

                        subi $t6, $t2, 1        # (i-1)
                        mul $t6, $t6, 4         # (i-1)x4

                        add $t5, $t5, $t6       # (j-1)xNx4 + (i-1)x4
                        add $t6, $a0, $t5       # t6 = array address + offset

                        lw $t1, ($t6)           #load the value we want to t1
```

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

        la $a0, tab
        li $v0, 4
        syscall

        addi $t3, $t3, 1
        move $a0, $s0

        j LPRCr

pC:
        bgt $t2, $a1, LPRCDone
        # returns the result in t5
        subi $t5, $t3, 1          # (j-1)
        mul $t5, $t5, $a1         # (j-1)xN
        mul $t5, $t5, 4           # (j-1)xNx4

        subi $t6, $t2, 1          # (i-1)
        mul $t6, $t6, 4           # (i-1)x4

        add $t5, $t5, $t6         # (j-1)xNx4 + (i-1)x4
        add $t6, $a0, $t5         # t6 = array address + offset

        lw $t1, ($t6)             #load the value we want to t1

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

        la $a0, tab
        li $v0, 4
        syscall

        addi $t2, $t2, 1
        move $a0, $s0

        j pC

LPRCDone:
```

```
                    li $t1, 0
                    li $t2, 0
                    li $t3, 0
                    li $t5, 0
                    li $t6, 0

                    jr $ra
#################################################

#
#

#
#             data segment

#
#

#
#

#################################################


       .data


msg110:     .asciiz "Welcome to the Lab3 program about linked lists.\n"


msg111:     .asciiz "Here are the options you can choose: \n"

msg112:     .asciiz "1 - enter N for matrix dimensions \n"

msg113:     .asciiz "2 - Display WHOLE matrix \n"

msg114:     .asciiz "3 - Obtain summation of matrix elements row-major (row by row)
summation \n"

msg115:     .asciiz "4 - Obtain summation of matrix elements column-major (column
by column) summation  \n"
```

```
msg116:		.asciiz "5 - Display desired elements of the matrix by specifying its row and column member \n"

msg116.2: .asciiz "6 - Display row or column (Menu Item 3 depending on the lab paper) \n"

msg118:		.asciiz "7 - exit this program \n"

msg6.1:		.asciiz "1 - row \n"
msg6.2:		.asciiz "2 - column \n"
msg6.3:		.asciiz "Enter index for row or column: "

msg119:		.asciiz "Enter the integer for the action you choose:  "

msg1.1:		.asciiz "Enter N: "
msg5.1:		.asciiz "Enter i: "
msg5.2:		.asciiz "Enter j: "

tab:	.asciiz "\t"



msg127:		.asciiz "Thanks for using the Lab6 program about matrix.\n"



msg128:		.asciiz "You must enter an integer from 1 to 6. \n"



endl:		.asciiz "\n"


##


## end of file matrix
```