CS 224

Section.: 2

Spring 2019

Lab No.: 6

Zeynep Cankara/ 21703381

Question 1:

No.	Cache Size KB	N way Cache	Word Size (bits)	Block Size	No of Sets	Tag Size (bits)	Index Size (set no)	Word block offset	Byte offset size	Block replacement Needed?
1	32	1	32	4	2^{11}	17	11	2	2	No
2	32	2	32	4	2^{10}	18	10	2	2	Yes
3	32	4	32	8	2^{8}	19	8	3	2	Yes
4	32	Full	32	8	2^{0}	27	0	3	2	Yes
9	256	1	16	4	2^{15}	14	15	2	1	No
10	256	2	16	4	2^{14}	15	14	2	1	Yes
11	256	4	8	16	2^{12}	16	12	4	0	Yes
12	256	Full	8	16	2^{0}	28	0	4	0	Yes

Question 2:

a.

Instruction	Iteration No.				
	1	2	3	4	5
lw $t1, 0x24($0)	Compulsory	Hit	Hit	Hit	Hit
lw $t2, 0x2C($0)	Compulsory	Hit	Hit	Hit	Hit
lw $t3, 0x28($0)	Hit	Hit	Hit	Hit	Hit

b.

Cache Representation

V	Tag	Data	Data	
1(bit)	(27 bits)	(32 bits)	(32 bits)	set 3
1(bit)	(27 bits)	(32 bits)	(32 bits)	set 2
1(bit)	(27 bits)	(32 bits)	(32 bits)	set 1
1(bit)	(27 bits)	(32 bits)	(32 bits)	set 0

Cache Capacity: 8 words

Block Size: 2 Words

N = 1 (direct associative cache)

Byte offset: 2 bit | Set: 2 bit | Block offset: 1 bit| Tag = 32 - (2+2+1) = 27 bits

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

c.

Hardware Required for the proposed cache design.

1 2:1 MULTIPLEXER for selecting the word within the block

1 EQUALITY COMPARATOR for checking tag matching

1 AND gate to determine the hit

Question 3:

a.

Note: Because capacity is full it in turn cause conflicts.

Instruction	Iteration No.				
	1	2	3	4	5
lw $t1, 0x24($0)	Compulsory	Capacity	Capacity	Capacity	Capacity
lw $t2, 0x2C($0)	Compulsory	Capacity	Capacity	Capacity	Capacity
lw $t3, 0x28($0)	Capacity	Capacity	Capacity	Capacity	Capacity

b.

Cache Representation

V	Tag	Data	V	Tag	Data	
(1 bit)	(30 bits)	(32 bits)	(1 bit)	(30 bits)	(32 bits)	Set

Cache Capacity: 2 words

Block Size: 1 word

N = 2

Byte offset: 2 bit | Set: 0 bit | Block offset: 0 bit| Tag = 32 - (2) = 30 bits

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

c.

Hardware Required for the proposed cache design.

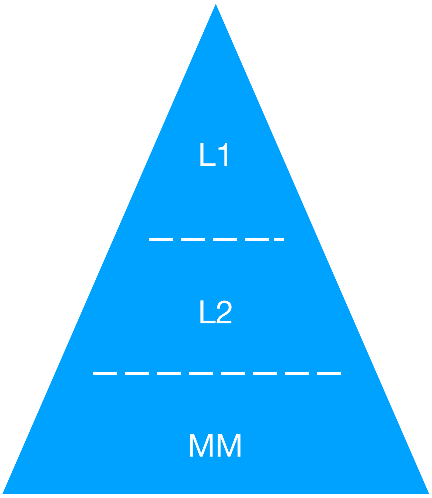
1 2:1 MULTIPLEXER for selecting the way within the block

2 EQUALITY COMPARATOR for checking tag matching

2 AND and 1 OR gates to determine the hit

Question 4:

Illustration of Memory Hierarchy :



Goal: Calculate AMAT in number of clock cycles

T_{L1} = 1 clock cycles

T_{L2} = 4 clock cycles

T_{MM} = 80 clock cycles

$Miss_{L1}$ = 5% miss rate L1

$Miss_{L2}$ = 25% miss rate L2

$AMAT = T_{L1} + Miss_{L1} . (T_{L2} + Miss_{L2} . T_{MM})$

$= 1 + 0.05 \times (4 + 80 \times 0.25)$
$= 2.2$ clock cycles

Clock rate with 2GHz = 0.5 ns

10^{12} Instructions requires time $= 10^{12} \times 0.5 \times 2.2$
$= 1.1 \times 10^{12}$ ns
$= 1100$ s

Question 5:

```
# ========== TEXT SECTION ============ ###
           .text
      .globl     __start

__start:
      jal main # interactive menu
      li $v0, 10 # exit the programme
      syscall

# interactive menu
# WORKING [+]
main:
      # save return address to the stack
      addi $sp, $sp, -4
      sw $ra, 0($sp)
      # Display intro message
      li $v0, 4
      la $a0, prompt # Prompt the assignment Task
      syscall
      menu:
           # Call display
           jal display
           # Ask user to enter option
           li $v0, 4
           la $a0, msgOption
           syscall
           #user option selection
```

```
  li $v0, 5 # v0 read integer
  syscall
  # Call sub programs according to options
  beq $v0, 1, opt1
  beq $v0, 2, opt2
  beq $v0, 3, opt3
  beq $v0, 4, opt4
  beq $v0, 5, opt5
beq $v0, 6, opt6
beq $v0, 7, opt7
beq $v0, 8, opt8
beq $v0, 9, optExit
opt1:
  li $v0, 4
        la $a0, msgN # Message to read N (NxN matrix)
        syscall
        li $v0, 5 # Read N
        syscall
        move $s0, $v0 # Store size in  -> $s0
        mul $s2, $s0, $s0 # Elements in NxN array -> $s2
        # Now allocate the Matrix space in heap
        mul $a0, $s2, 4 # bytes to allocate
        li $v0, 9 # dynamic memory allocation
        syscall # base address -> $v0
        move $s1, $v0 # base address matrix -> $s1
        # call sub program read values from user
        jal fillCustomMatrix
        j loopBack
  opt2:
        li $v0, 4
```

```
        la $a0, msgN # Message to read N (NxN matrix)
        syscall
        li $v0, 5 # Read N
        syscall
        move $s0, $v0 # Store size in  -> $s0
        mul $s2, $s0, $s0 # Elements in NxN array -> $s2
        # Now allocate the Matrix space in heap
        mul $a0, $s2, 4 # bytes to allocate
        li $v0, 9 # dynamic memory allocation
        syscall # base address -> $v0
        move $s1, $v0 # base address matrix -> $s1
        # call sub program fill matrix with consecutive elements
        jal fillConsecutiveMatrix
        j loopBack
opt3:
        # takes row and column and displays the element
        li $v0, 4 # request to read row  ([i], j)
        la $a0, msgRow
        syscall
        li $v0, 5 # read row i
        syscall
        move $t0, $v0 # row -> $t0
        li $v0, 4 # request to read col (i, [j])
        la $a0, msgCol
        syscall
        li $v0, 5 # read col j
        syscall
        move $t1, $v0 # col -> $t1
        # calculate the position
        subi $t0, $t0, 1 # (i - 1) -> $t0
```

```
        mul $t0, $t0, $s0 # (i - 1) * N -> $t0
        mul $t0, $t0, 4 # (i - 1) * N * 4 -> $t0
        subi $t1, $t1, 1 # (j - 1) -> $t1
        mul $t1, $t1, 4 # (j - 1) * 4 -> $t1
        add $t0, $t0, $t1 # (i - 1) * N * 4 + (j - 1) * 4 -> $t0
        add $t2, $t0, $s1 # effective address of the position
        # display result prompt
        li $v0, 4
        la $a0, msgOpt3Res # result
        syscall
        # display the item
        lw $a0, 0($t2)
        li $v0, 1
        syscall
        j loopBack
opt4:
        # display the matrix row by row
        jal displayMatrix
        j loopBack
opt5:
        # Find the trace of a matrix
        jal findTrace
        j loopBack
opt6:
        # Find the trace like matrix
        jal findTraceLike
        j loopBack
opt7:
        # obtain row by row summation
        jal sumRowWise
```

```
            j loopBack
        opt8:
            # obtain col by col summation
            jal sumColWise
            j loopBack
       # execute main loop again
       loopBack:
        j menu # go back to menu
       optExit: # exit the loop
    # load back the return address
    lw $ra, 0($sp)
    addi $sp, $sp, 4
    jr $ra # go to

# Finds the trace like summation of the matrix
findTraceLike:
    addi $sp, $sp, -16 # malloc
    sw $ra, 0($sp)
    sw $s1, 4($sp)
    sw $s2, 8($sp)
    sw $s0, 12($sp)
    # =================
    li $a0, 1
    addi $a1, $s0, 0
    li $t4, 0 # trace like summation -> $t4
    traceLoop2:
        # call sub-routine to obtain trace like summation
        jal getItem
        add $t4, $t4, $v0
        addi $a0, $a0, 1
```

```
        subi $a1, $a1, 1
        ble $a0, $s0, traceLoop2
    # display the result prompt
    li $v0, 4
    la $a0, msgOpt6Res # result
    syscall
    # display the trace like summation
    move $a0, $t4
    li $v0, 1
    syscall
    # =================
    # calloc
    lw $s0, 12($sp)
    lw $s2, 8($sp)
    lw $s1, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 16
    jr $ra # goto


# Get Item
# Returns item at the position
# row -> $a0
# col -> $a0
# item -> $v0
# WORKING [+]
getItem:
    addi $sp, $sp, -16 # malloc
    sw $ra, 0($sp)
    sw $s1, 4($sp)
```

```
	sw $s2, 8($sp)
	sw $s0, 12($sp)
	# =================
	move $t0, $a0 # row -> $t0
	move $t1, $a1 # col -> $t1
	# calculate the position
	subi $t0, $t0, 1 # (i - 1) -> $t0
	mul $t0, $t0, $s0 # (i - 1) * N -> $t0
	mul $t0, $t0, 4 # (i - 1) * N * 4 -> $t0
	subi $t1, $t1, 1 # (j - 1) -> $t1
	mul $t1, $t1, 4 # (j - 1) * 4 -> $t1
	add $t0, $t0, $t1 # (i - 1) * N * 4 + (j - 1) * 4 -> $t0
	add $t2, $t0, $s1 # effective address of the position
	# fetch the item
	lw $v0, 0($t2)
	# =================
	# calloc
	lw $s0, 12($sp)
	lw $s2, 8($sp)
	lw $s1, 4($sp)
	lw $ra, 0($sp)
	addi $sp, $sp, 16
	jr $ra # goto

# Find trace of the matrix display
# WORKING [+]
findTrace:
	addi $sp, $sp, -16 # malloc
	sw $ra, 0($sp)
	sw $s1, 4($sp)
```

```
    sw $s2, 8($sp)
    sw $s0, 12($sp)
    # =================
    li $a0, 1
    li $a1, 1
    li $t4, 0 # trace -> $t4
    traceLoop:
        jal getItem
        add $t4, $t4, $v0
        addi $a0, $a0, 1
        addi $a1, $a1, 1
        ble $a0, $s0, traceLoop
    # display the result prompt
    li $v0, 4
    la $a0, msgOpt5Res # result
    syscall
    # display the trace
    move $a0, $t4
    li $v0, 1
    syscall
    # =================
    # calloc
    lw $s0, 12($sp)
    lw $s2, 8($sp)
    lw $s1, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 16
    jr $ra # goto

# Displays matrix row by row
```

```
# WORKING [+]
displayMatrix:
    addi $sp, $sp, -12 # malloc
    sw $ra, 0($sp)
    sw $s1, 4($sp)
    sw $s2, 8($sp)
    # =================
    li $v0, 4 # matrix construction
    la $a0, endl
   syscall
   li $t0, 2 # use to control end of line (endl)
    sumRowLoop2:
        lw $a0, 0($s1) # current item -> $a0
        li $v0, 1
        syscall # print current element
        li $v0, 4 # matrix construction
    la $a0, wSpace
      syscall
        addi $s1, $s1, 4 # iterate matrix
        subi $s2, $s2, 1
        ble $t0, $s0, jEnter
        li $t0, 1 # use to control end of line (endl)
        li $v0, 4 # matrix construction
    la $a0, endl
      syscall
        jEnter:
        addi $t0, $t0, 1
        bgt $s2, $0, sumRowLoop2
    # calloc
    lw $s2, 8($sp)
```

```
	lw $s1, 4($sp)
	lw $ra, 0($sp)
	addi $sp, $sp, 12
	jr $ra # goto

# Obtains col by col summations of elements within the matrix
# Column-major representation
# WORKING [+]
sumColWise:
	addi $sp, $sp, -12 # malloc
	sw $ra, 0($sp)
	sw $s1, 4($sp)
	sw $s2, 8($sp)
	# =======
	mul $t3, $s0, 4 # row offset -> $t3
	li $t0, 0 # sum -> $t0
	li $t1, 0 # (col - 1) -> $t1
	coLoopNextRow:
		mul $t4, $t1, 4 # col offset -> $t4
		move $t2, $0 # row - 1 -> $t2
		add $t5, $s1, $t4 # effective memory address -> $t5
		lw $t7, 0($t5) # current item -> $t7
		add $t0, $t7, $t0 # recalculate sum
		cLoopNextCol:
			add $t5, $t3, $t5 # effective memory address -> $t5
			lw $t7, 0($t5) # current item -> $t7
			add $t0, $t7, $t0 # recalculate sum
			addi $t2, $t2, 1 # row++
			blt $t2, $s0, cLoopNextCol # row < N keep continue
		addi $t1, $t1, 1 # col++
```

```
        blt $t1, $s0, coLoopNextRow
    # display result
    li $v0, 4 # result message
    la $a0, msgOpt8Res
    syscall
    # print the result
    addi $a0, $t0, 0
    li $v0, 1
    syscall
    # =======
    # calloc
    lw $s2, 8($sp)
    lw $s1, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 12
    jr $ra # goto

# Obtains row by row summations of elements within the matrix
# Row-major representation
# WORKING [+]
sumRowWise:
    addi $sp, $sp, -12 # malloc
    sw $ra, 0($sp)
    sw $s1, 4($sp)
    sw $s2, 8($sp)
    li $t0, 0 # sum -> $t0
    move $t2, $s2 # Counter for items
    sumRowLoop:
        lw $t1, 0($s1) # current item -> $t1
        addi $s1, $s1, 4 #iterate over the matrix
```

```
        add $t0, $t0, $t1 # sum -> $t0
        subi $t2, $t2, 1
        bgt $t2, $0, sumRowLoop
    li $v0, 4 # result message
    la $a0, msgOpt7Res
    syscall
    # print the result
    addi $a0, $t0, 0
    li $v0, 1
    syscall
    # calloc
    lw $s2, 8($sp)
    lw $s1, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 12
    jr $ra # goto

# Fills the matrix with consecutive integers
# WORKING [+]
fillConsecutiveMatrix:
    addi $sp, $sp, -12 # matrix base and return address saved
    sw $ra, 0($sp)
    sw $s1, 4($sp)
    sw $s2, 8($sp)
    # start filling from 1
    li $t1, 1 # item value -> $t1
    writeItems:
        sw $t1, 0($s1) # write to the array
        addi $s1, $s1, 4 # next item of the matrix
        addi $t1, $t1, 1 # increment element
```

```
        sle  $t3, $t1, $s2
        beq $t3, 1, writeItems
    lw $s2, 8($sp)
    lw $s1, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 12
    jr $ra # goto

# Fills the matrix with user defined values
# WORKING [+]
fillCustomMatrix:
    addi $sp, $sp, -12 # matrix base and return address saved
    sw $ra, 0($sp)
    sw $s1, 4($sp)
    sw $s2, 8($sp)
    # start filling from 1
    li $t1, 1 # iterator -> $t1
    writeItems2:
        # read items from the user
        li $v0, 5
        syscall
        move $t0, $v0 # user input -> $t0
        sw $t0, 0($s1) # write to the array
        addi $s1, $s1, 4 # next item of the matrix
        addi $t1, $t1, 1 # increment element
        ble $t1, $s2, writeItems2
    lw $s2, 8($sp)
    lw $s1, 4($sp)
    lw $ra, 0($sp)
    addi $sp, $sp, 12
```

```
    jr $ra # goto

# Menu display interface
# WORKING [+]
display:
    addi $sp, $sp, -4 # allocate stack space
    sw $ra, 0($sp)
    li $v0, 4
    # print the options
    la $a0, msgMenu
    syscall
    la $a0, msgOpt1
    syscall
    la $a0, msgOpt2
    syscall
    la $a0, msgOpt3
    syscall
    la $a0, msgOpt4
    syscall
    la $a0, msgOpt5
    syscall
    la $a0, msgOpt6
    syscall
    la $a0, msgOpt7
    syscall
    la $a0, msgOpt8
    syscall
    la $a0, msgExitOpt
    syscall
    lw $ra, 0($sp) # goto return address
```

```
      addi $sp, $sp, 4

      jr $ra # goto


# ========== DATA SECTION ============ ###

            .data

prompt:         .asciiz     "Interactive menu to perform operations on
an user defined matrix \n"

msgOption: .asciiz   "\n Please choose an option: "

msgMenu: .asciiz "\n =========== MENU =============="

msgOpt1: .asciiz "\n  1. Create matrix (NxN) with user values"

msgOpt2: .asciiz "\n  2. Create matrix (NxN) with consecutive values"

msgOpt3: .asciiz    "\n  3. Display the target item (row, col)"

msgOpt4: .asciiz    "\n  4. Display matrix (Row by row)"

msgOpt5: .asciiz    "\n  5. Obtain trace of the matrix -> display"

msgOpt6: .asciiz    "\n  6. Obtain trace like summation (other diagonal)
-> display"

msgOpt7: .asciiz    "\n  7. Obtain sum of matrix elements (Row by row)"

msgOpt8: .asciiz    "\n  8. Obtain sum of matrix element (Column by
column) "

msgExitOpt: .asciiz "\n  9. Exit"

msgN: .asciiz "\n Please enter N of (NxN) matrix: "

msgRow: .asciiz "\n Please enter Row number i [1:N]: "

msgCol: .asciiz "\n Please enter Column number j [1:N]: "

msgOpt3Res: .asciiz "\n Item in the given row & col: "

msgOpt7Res: .asciiz "\n Sum of elements obtained in terms of Row-Major
iteration (row by row): "

msgOpt8Res: .asciiz "\n Sum of elements obtained in terms of Column-Major
iteration (col by col): "

msgOpt5Res: .asciiz  "\n Trace of the NxN matrix: "

msgOpt6Res: .asciiz  "\n Trace like summation of the NxN matrix: "

endl: .asciiz "\n"

wSpace: .asciiz " "
```