CS224
Section No: 1
Fall 2019
Lab No 6
Fatih Sevban Uyanik / 21602486

Q1) **Fill in the empty cells of the following table. Assume that main memory size is 4GB. Index Size: No. of bits needed to express the set number in an address, Block Offset: No. of bits needed to indicate the word offset in a block, Byte Offset: No. of bits needed to indicate the byte offset in a word. Block Replacement Policy Needed: Indicate if a block replacement policy such as FIFO, LRU, LFU etc. is needed (yes) or not (no). If some combinations are not possible mark them.**

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	2^0	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	2^0	27	0	4	1	YES

Q2) Consider the following MIPS code segment. Cache capacity is 8 words, Block size: 2 words, N= 1.

```
      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.

1 bit V is required. Data section consists of 32 bit and there are two data sections. 27 bit tag is present. As a result, 92 bits constitute one block. There are 4 blocks. Multiplying 92 bits with 4, **368 bits** is obtained.

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

And Gate ---> 1
Equality Comparator ---> 1
2x1 Mux ---> 1
is needed for implementation.

Q3) 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.

For the implementation of LRU, 1bit is needed for the cache. 1 bit is for V section and there are two V sections. 30 bits is for tag and 2 tag sections are present. Data consists of 32 bits and again, there are two data sections in the cache. As a result, the total cache memory is as the following:

(30 + 32 +1) * 2 + 1 = <u>127 bits</u>

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

And Gate ---> 2
Equality Comparator ---> 2
2x1 Mux ---> 1
Or Gate ---> 1
is needed for implementation.

Q4) **Consider a three-level memory: L1 and L2 are for cache memory and the third level is for the main memory. Access time for L1 is 1 clock cycle, the access time for L2 is 4 times more than L1 and main memory access time is 10 times more than L2. The miss rate for L1 is 20% and the miss rate for L2 is 5%. What is the effective clock cycle for memory access (AMAT in number of clock cycles)?**

Access Times
L1 Cache ---> 1 clock cycle
L2 Cache ---> 4 clock cycle
Main memory ---> 40 cycle

AMAT --> 1 + 0.20 * (4 + 0.05 * 40) = 2.2 cycles

With 4 GHz clock rate how much time is needed for a program with 10^{12} instructions to execute?

Execution time = (# instructions) * ($\frac{cycles}{instruction}$) * ($\frac{seconds}{cycle}$)
= 10^{12} * 2.2 * (0.25 * 10^{-9})
= 550 seconds

Q5)

```
.data
	option_1_create_matrix:		.asciiz "1 --> Create matrix\n"
	option_2_retrieve_item:		.asciiz "2 --> Retrieve Item\n"
	option_3_row_by_row_sum:	.asciiz "3 --> Row by row Sum\n"
	option_4_col_by_col_sum:	.asciiz "4 --> Col by col Sum\n"
	option_5_display_matrix:	.asciiz "5 --> Display Matrix\n"
	option_6_display_row_col:	.asciiz "6 --> Display row and column\n"
	option_7_create_automatic: .asciiz "7 --> Create matrix and fill automatic\n"
	option_8_exit: .asciiz "8 --> Exit\n"
	request_matrix_size: .asciiz "Enter matrix size: "
	request_entry: .asciiz "Please enter item at "
	request_option: .asciiz "Please select an option: "
	request_row: .asciiz "Please select row index: "
	request_col: .asciiz "Please select col index: "
	paranthesis_left: .asciiz "("
	paranthesis_right: .asciiz ")"
	comma: .asciiz ","
	row: .asciiz "ROW: "
	col: .asciiz "COL: "
	space: .asciiz " "
	two_dots: ": "
	new_line: .asciiz "\n"
	divider: .asciiz "--------------------------------------------------------------\n"
.text
	main:
		while:
			jal print_interface

			la $a0, request_option
			li $v0, 4
			syscall

			# getting user input
			addi $v0, $zero, 5
			syscall
			move $s6, $v0

			option_1: bne $s6, 1, option_2
			jal create_matrix
			move $s0, $v0 # size of matrix
			move $s1, $v1 # address of matrix

			option_2: bne $s6, 2, option_3
			move $a0, $s0
			move $a1, $s1

			la $a0, request_row
			li $v0, 4
			syscall

			# getting user input
			addi $v0, $zero, 5
			syscall
			move $a2, $v0

			la $a0, request_col
			li $v0, 4
			syscall

			# getting user input
			addi $v0, $zero, 5
```

```
			syscall
			move $a3, $v0
			jal retrieve_item

			option_3: bne $s6, 3, option_4
			move $a0, $s0
			move $a1, $s1
			jal sum_row_by_row

			option_4: bne $s6, 4, option_5
			move $a0, $s0
			move $a1, $s1
			jal sum_col_by_col

			option_5: bne $s6, 5, option_6
			move $a0, $s0
			move $a1, $s1
			jal display_matrix

			option_6: bne $s6, 6, option_7
			la $a0, request_row
			li $v0, 4
			syscall
			move $a2, $v0

			# getting user input
			addi $v0, $zero, 5
			syscall
			move $a2, $v0

			la $a0, request_col
			li $v0, 4
			syscall

			# getting user input
			addi $v0, $zero, 5
			syscall
			move $a3, $v0

			move $a0, $s0
			move $a1, $s1
			jal display_row_and_col

			option_7: bne $s6, 7, option_8
			jal create_matrix_automatic
			move $s0, $v0 # size of matrix
			move $s1, $v1 # address of matrix

			option_8: beq $s6, 8, exit
			j while
		exit:

# main
li $v0, 10
syscall

print_interface:
		la $a0, divider
		li $v0, 4
		syscall

		la $a0, option_1_create_matrix
		li $v0, 4
```

```
	syscall

	la $a0, option_2_retrieve_item
	li $v0, 4
	syscall

	la $a0, option_3_row_by_row_sum
	li $v0, 4
	syscall

	la $a0, option_4_col_by_col_sum
	li $v0, 4
	syscall

	la $a0, option_5_display_matrix
	li $v0, 4
	syscall

	la $a0, option_6_display_row_col
	li $v0, 4
	syscall

	la $a0, option_7_create_automatic
	li $v0, 4
	syscall

	la $a0, option_8_exit
	li $v0, 4
	syscall

	la $a0, divider
	li $v0, 4
	syscall
# print_interface
jr $ra


display_row_and_col:
addi $sp, $sp, -32
sw $ra, 0($sp)
sw $s0, 4($sp)
sw $s1, 8($sp)
sw $s2, 12($sp)
sw $s3, 16($sp)
sw $s4, 20($sp)
sw $s5, 24($sp)
sw $s6, 28($sp)

	move $s0, $a0 # size
	move $s1, $a1 # address of array
	move $s2, $a2 # row
	move $s3, $a3 # col

	mul $t0, $s0, 4
	mul $t0, $t0, $s2
	add $t1, $t0, $s1
	move $s4, $zero

	la $a0, row
	li $v0, 4
	syscall

	loop_9: beq $s4, $s0, exit_9
```

```
            lw $s5, 0($t1)

            move $a0, $s5
            li $v0, 1
            syscall

            la $a0, space
            li $v0, 4
            syscall

            addi $s4, $s4, 1
            addi $t1, $t1, 4
            j loop_9
            exit_9:

        la $a0, new_line
        li $v0, 4
        syscall

        mul $s5, $s0, 4
        mul $t0, $s3, 4
        add $t0, $t0, $s1
        move $s4, $zero

        la $a0, col
        li $v0, 4
        syscall

        loop_10: beq $s4, $s0, exit_10
             lw $s6, 0($t0)

             move $a0, $s6
             li $v0, 1
             syscall

             la $a0, space
             li $v0, 4
             syscall

             addi $s4, $s4, 1
             add $t0, $t0, $s5
             j loop_10
             exit_10:

        la $a0, new_line
        li $v0, 4
        syscall

lw $ra, 0($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
lw $s4, 20($sp)
lw $s5, 24($sp)
lw $s6, 28($sp)
addi $sp, $sp, 32
jr $ra
# sum_row_by_row


retrieve_item:
addi $sp, $sp, -32
```

```
sw $ra, 0($sp)
sw $s0, 4($sp)
sw $s1, 8($sp)
sw $s2, 12($sp)
sw $s3, 16($sp)
sw $s4, 20($sp)
sw $s5, 24($sp)
sw $s6, 28($sp)

        move $s0, $a0 # size
        move $s1, $a1 # address of array
        move $s2, $a2 # row
        move $s3, $a3 # col

        mul $t0, $s0, 4
        mul $t0, $t0, $s2
        mul $t1, $s3, 4
        add $t0, $t0, $t1
        add $t1, $t0, $s1
        lw $s4, 0($t1)

        move $a0, $s4
        li $v0, 1
        syscall

lw $ra, 0($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
lw $s4, 20($sp)
lw $s5, 24($sp)
lw $s6, 28($sp)
addi $sp, $sp, 32
jr $ra


sum_col_by_col:
addi $sp, $sp, -32
sw $ra, 0($sp)
sw $s0, 4($sp)
sw $s1, 8($sp)
sw $s2, 12($sp)
sw $s3, 16($sp)
sw $s4, 20($sp)
sw $s5, 24($sp)
sw $s6, 28($sp)

        move $s0, $a0 # size
        move $s1, $a1 # address of array
        move $s2, $zero
        move $s3, $zero

        loop_7: beq $s3, $s0, exit_7
                move $s6, $zero
                loop_8: beq $s2, $s0, exit_8
                        mul $t0, $s0, 4
                        mul $t0, $t0, $s2
                        mul $t1, $s3, 4
                        add $t0, $t0, $t1
                        add $t1, $t0, $s1
                        lw $s4, 0($t1)
                        add $s6, $s6, $s4
```

```
                addi $s2, $s2, 1
            j loop_8
            exit_8:

            move $s2, $zero
            addi $s3, $s3, 1

            move $a0, $s6
            li $v0, 1
            syscall

            la $a0, space
            li $v0, 4
            syscall
            j loop_7
            exit_7:

            la $a0, new_line
            li $v0, 4
            syscall

lw $ra, 0($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
lw $s4, 20($sp)
lw $s5, 24($sp)
lw $s6, 28($sp)
addi $sp, $sp, 32
jr $ra
# sum_col_by_col


sum_row_by_row:
addi $sp, $sp, -32
sw $ra, 0($sp)
sw $s0, 4($sp)
sw $s1, 8($sp)
sw $s2, 12($sp)
sw $s3, 16($sp)
sw $s4, 20($sp)
sw $s5, 24($sp)
sw $s6, 28($sp)

        move $s0, $a0 # size
        move $s1, $a1 # address of array
        move $s2, $zero
        move $s3, $zero

        loop_1: beq $s2, $s0, exit_1
            move $s6, $zero
            loop_2: beq $s3, $s0, exit_2
                mul $t0, $s0, 4
                mul $t0, $t0, $s2
                mul $t1, $s3, 4
                add $t0, $t0, $t1
                add $t1, $t0, $s1
                lw $s4, 0($t1)
                add $s6, $s6, $s4
                addi $s3, $s3, 1
            j loop_2
            exit_2:
```

```
                move $s3, $zero
                addi $s2, $s2, 1

                move $a0, $s6
                li $v0, 1
                syscall

                la $a0, new_line
                li $v0, 4
                syscall
                j loop_1
                exit_1:

lw $ra, 0($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
lw $s4, 20($sp)
lw $s5, 24($sp)
lw $s6, 28($sp)
addi $sp, $sp, 32
jr $ra
# sum_row_by_row


display_matrix:
addi $sp, $sp, -32
sw $ra, 0($sp)
sw $s0, 4($sp)
sw $s1, 8($sp)
sw $s2, 12($sp)
sw $s3, 16($sp)
sw $s4, 20($sp)
sw $s5, 24($sp)
sw $s6, 28($sp)

        move $s0, $a0 # size
        move $s1, $a1 # address of array
        move $s2, $zero
        move $s3, $zero

        loop_5: beq $s2, $s0, exit_5
                loop_6: beq $s3, $s0, exit_6

                        mul $t0, $s0, 4
                        mul $t0, $t0, $s2
                        mul $t1, $s3, 4
                        add $t0, $t0, $t1
                        add $t1, $t0, $s1
                        lw $s4, 0($t1)

                        move $a0, $s4
                        li $v0, 1
                        syscall

                        li $v0, 4
                        la $a0, space
                        syscall
                        addi $s3, $s3, 1
                j loop_6
                exit_6:
```

```
            move $s3, $zero
            addi $s2, $s2, 1
            la $a0, new_line
            li $v0, 4
            syscall
            j loop_5
            exit_5:

lw $ra, 0($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
lw $s4, 20($sp)
lw $s5, 24($sp)
lw $s6, 28($sp)
addi $sp, $sp, 32
jr $ra
# display_matrix


create_matrix:
addi $sp, $sp, -32
sw $ra, 0($sp)
sw $s0, 4($sp)
sw $s1, 8($sp)
sw $s2, 12($sp)
sw $s3, 16($sp)
sw $s4, 20($sp)
sw $s5, 24($sp)
sw $s6, 28($sp)

      li $v0, 4
      la $a0, request_matrix_size
      syscall

      # getting user input
      addi $v0, $zero, 5
      syscall

      move $s0, $v0
      mul $s5, $s0, $s0
      mul $s5, $s5, 4

      move $a0, $s5
      li $v0, 9
      syscall

      move $s1, $v0
      move $s2, $zero
      move $s3, $zero

      loop_3: beq $s2, $s0, exit_3
            loop_4: beq $s3, $s0, exit_4
                  la $a0, request_entry
                  li $v0, 4
                  syscall

                  beq $s3, $s0, exit_2
                  la $a0, paranthesis_left
                  li $v0, 4
                  syscall
```

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

            li $v0, 4
            la $a0, comma
            syscall

            move $a0, $s3
            li $v0, 1
            syscall

            li $v0, 4
            la $a0, paranthesis_right
            syscall

            li $v0, 4
            la $a0, two_dots
            syscall

            # getting user input
            addi $v0, $zero, 5
            syscall

            move $s4, $v0

            mul $t0, $s0, 4
            mul $t0, $t0, $s2
            mul $t1, $s3, 4
            add $t0, $t0, $t1
            add $t1, $t0, $s1
            sw $s4, 0($t1)
            addi $s3, $s3, 1

        j loop_4
        exit_4:

        move $s3, $zero
        addi $s2, $s2, 1
        j loop_3
        exit_3:

        move $v0, $s0
        move $v1, $s1
lw $ra, 0($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
lw $s4, 20($sp)
lw $s5, 24($sp)
lw $s6, 28($sp)
addi $sp, $sp, 32
jr $ra
# create_matrix

create_matrix_automatic:
addi $sp, $sp, -32
sw $ra, 0($sp)
sw $s0, 4($sp)
sw $s1, 8($sp)
sw $s2, 12($sp)
```

```
sw $s3, 16($sp)
sw $s4, 20($sp)
sw $s5, 24($sp)
sw $s6, 28($sp)

        li $v0, 4
        la $a0, request_matrix_size
        syscall

        # getting user input
        addi $v0, $zero, 5
        syscall

        move $s0, $v0
        mul $s5, $s0, $s0
        mul $s5, $s5, 4

        move $a0, $s5
        li $v0, 9
        syscall

        move $s1, $v0
        move $s2, $zero
        move $s3, $zero
        move $s4, $zero

        loop_11: beq $s2, $s0, exit_11
                loop_12: beq $s3, $s0, exit_12
                        mul $t0, $s0, 4
                        mul $t0, $t0, $s2
                        mul $t1, $s3, 4
                        add $t0, $t0, $t1
                        add $t1, $t0, $s1
                        sw $s4, 0($t1)
                        addi $s3, $s3, 1
                        addi $s4, $s4, 1
                j loop_12
                exit_12:

                move $s3, $zero
                addi $s2, $s2, 1
                j loop_11
                exit_11:

                move $v0, $s0
                move $v1, $s1
lw $ra, 0($sp)
lw $s0, 4($sp)
lw $s1, 8($sp)
lw $s2, 12($sp)
lw $s3, 16($sp)
lw $s4, 20($sp)
lw $s5, 24($sp)
lw $s6, 28($sp)
addi $sp, $sp, 32
jr $ra
```