**Atul Pande**

**2101CS88**

**CS210: (Lab 15)**

**Computer Architecture Lab**

**Cache Performance Analysis and Branch Prediction**

**Task 1: (30 points)**

The task is to examine the accuracy of various branch predictors for the following repeating patterns (e.g., in a loop) of branch outcomes. Accuracy is defined as the percentage of guesses that are correct.

**Consider C codes below and write MIPS assembly code for the same**

*(a)*

*Int i;*

*Int b[100];*

*Int a[100];*

*for(i=0; i< 100; i++) {*

*b[i] = a[a[i]];*

*}*

.data

array\_A: .word 2,1,4,3,6,5,8,7,10,9,12,11,14,13,16,15,18,17,20,19,22,21,24,23,26,25,28,27,30,29,32,31,34,33,36,35,38,37,40,39,42,41,44,43,46,45,48,47,50,49,52,51,54,53,56,55,58,57,60,59,62,61,64,63,66,65,68,67,70,69,72,71,74,73,76,75,78,77,80,79,82,81,84,83,86,85,88,87,90,89,92,91,94,93,96,95,98,97,100,99

array\_B: .word 0:100

.text

addiu $t0,$0,0

addiu $t2,$0,100

la $s0, array\_A

la $s1, array\_B

la $s2, array\_A

loop: slt $t1, $t0, $t2

beq $t1,$0,end

lw $t3, 0($s0)

addiu $t3,$t3,-1

mul $t3,$t3,4

addu $t3,$t3,$s2

lw $t3,0($t3)

sw $t3, 0($s1)

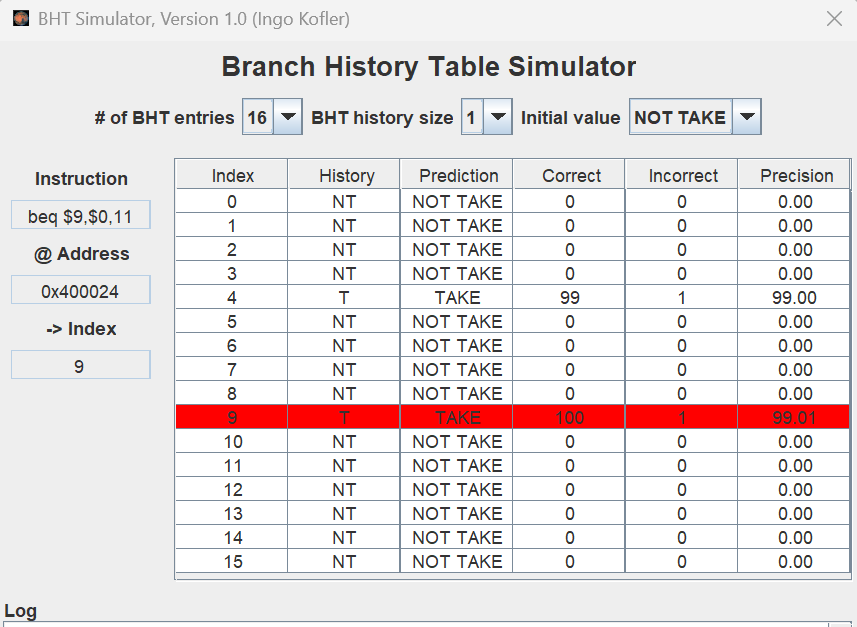
addiu $s0,$s0,4

addiu $s1,$s1,4

addiu $t0,$t0,1

beq $0,$0,loop

end:

****

**(b)**

**for(j = 0; j < 100; j++)**

**for(i = 0; i < 100; i++)**

**A[i] = i;**

.data

array\_A: .word 2,1,4,3,6,5,8,7,10,9,12,11,14,13,16,15,18,17,20,19,22,21,24,23,26,25,28,27,30,29,32,31,34,33,36,35,38,37,40,39,42,41,44,43,46,45,48,47,50,49,52,51,54,53,56,55,58,57,60,59,62,61,64,63,66,65,68,67,70,69,72,71,74,73,76,75,78,77,80,79,82,81,84,83,86,85,88,87,90,89,92,91,94,93,96,95,98,97,100,99

.text

addiu $t0,$0,0

addiu $t1,$0,0

addiu $t2,$0,100

la $s2,array\_A

outer\_loop: slt $s0,$t0,$t2

beq $s0,$0,end

inner\_loop: slt $s1,$t1,$t2

beq $t1,$t2,add1

sw $t1,0($s2)

addiu $s2,$s2,4

addiu $t1,$t1,1

beq $0,$0,inner\_loop

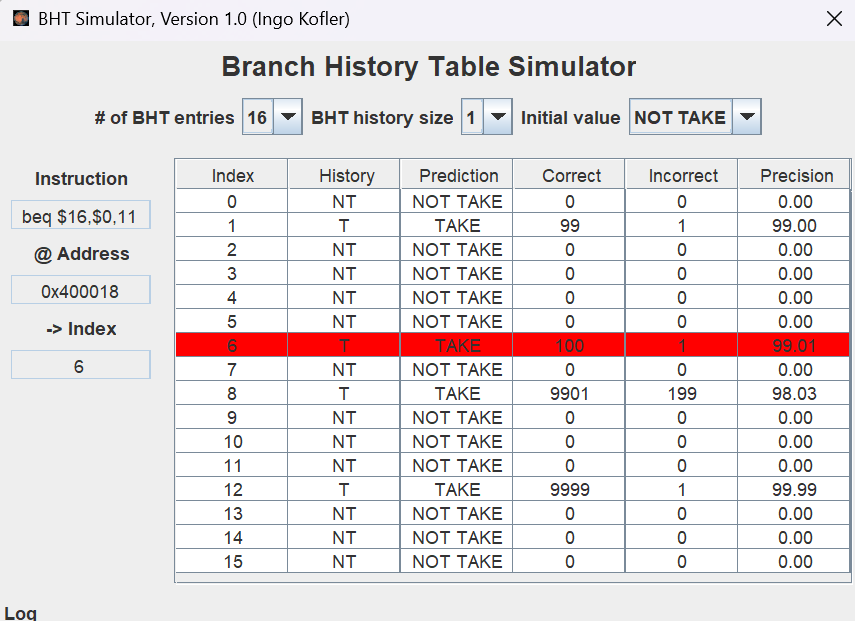
add1: addiu $t0,$t0,1

la $s2,array\_A

addiu $t1,$0,0

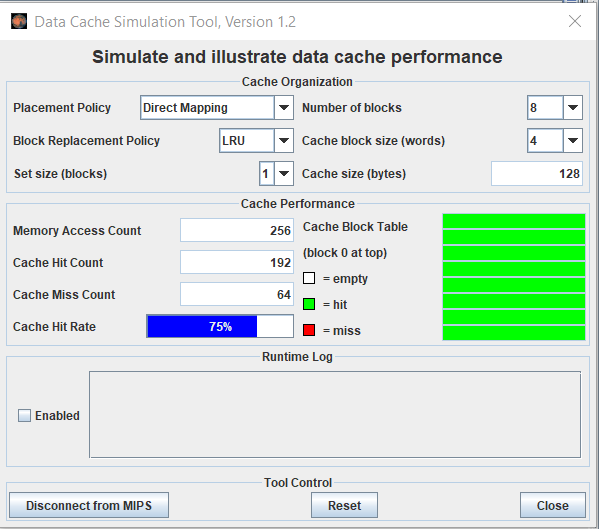
beq $0,$0,outer\_loop

end:

****

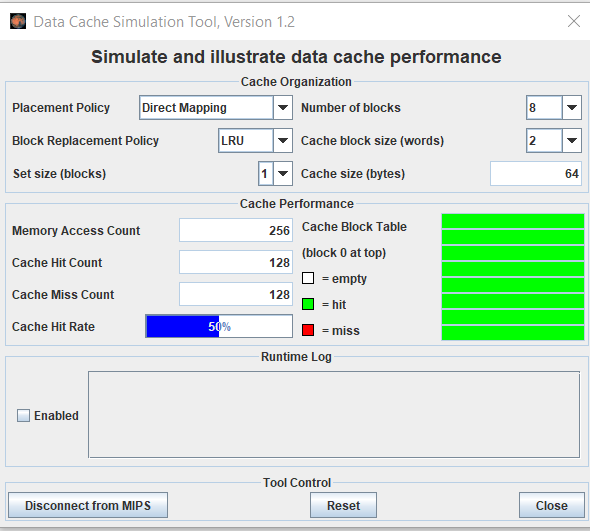
**Task 2: (25 points)**

**Complete the MARS tutorial Part 2.**

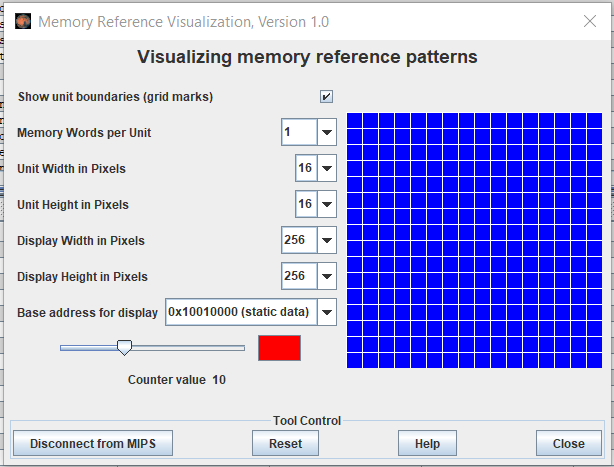
****

**Data Cache simulation for row\_major.asm ;**

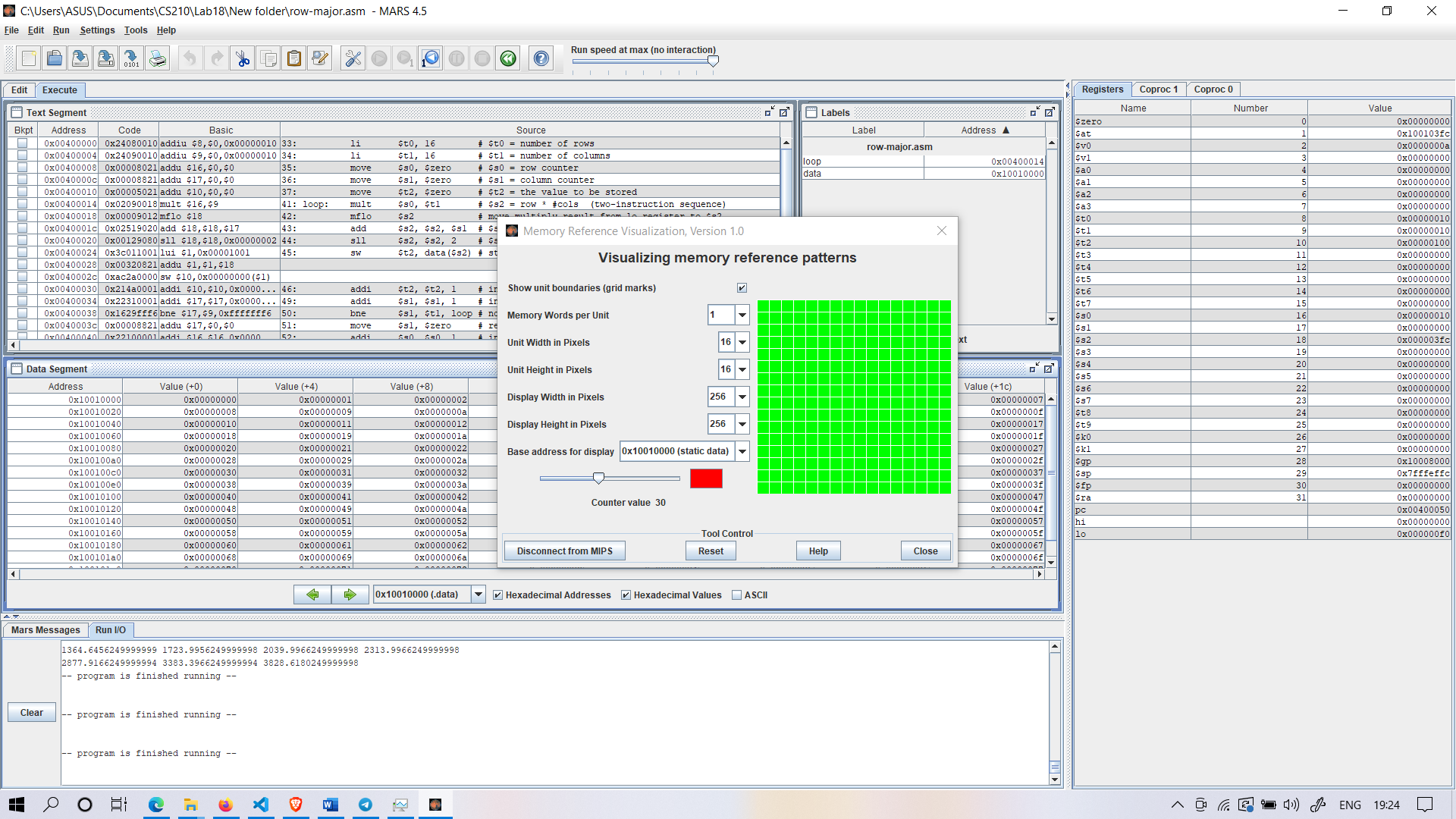
**Now on reducing block size**

****

**On reducing block size 2, hit rate decreases to 50% .**

****

**Row\_major memory reference visualtistaion . (at memory counter 10 )**

****

**Memory Reference At Counter Value 30**

Analysis:

The colour depends on the number of times the word has been referenced.

Black is 0, blue is 1, green is 2, yellow is 3 and 4, orange is 5 through 9, red is 10 or higher. In the first, memory is accessed rapidly than second counter value

**Task 3: (45 points)**

**Create two examples algorithms which demonstrate efficient use of data cache ( for example Matrix multiplication, ). Conduct performance analysis on different parameters such as block size, cache replacement algorithms.**

# Assume matrices A, B, and C are of size n x n

# Assume the base addresses of matrices A, B, and C are in $s0, $s1, and $s2, respectively

# Assume n is in $s3

# Assume the block size is in $s4

.data

A: .word 1 2 3 4 5 6 7 8 9

B: .word 1 0 0 0 1 0 0 0 1

C: .word 0 0 0 0 0 0 0 0 0

.text

la $s0, A

la $s1, B

la $s2, C

li $s3, 3

# Initialize loop counters

addi $t0, $zero, 0 # i = 0

addi $t1, $zero, 0 # j = 0

addi $t2, $zero, 0 # k = 0

loop\_i:

beq $t0, $s3, exit # Exit loop when i == n

li $t6, 12

mult $t6, $t0

mflo $t6

add $t7, $0, $t6

add $t6, $s0, $t6 # Calculate the starting address of row i of matrix A

add $t7, $s2, $t7 # Calculate the starting address of row i of matrix C

add $t1, $0, $0

loop\_j:

beq $t1, $s3, next\_j # Exit inner loop when j == n

sll $t8, $t1, 2

sll $t9, $t1, 2

add $t8, $s1, $t8 # Calculate the starting address of column j of matrix B

add $t9, $t7, $t9 # Calculate the address of C[i][j]

addi $t2, $zero, 0 # Reset k to 0

loop\_k:

beq $t2, $s3, next\_k # Exit innermost loop when k == n

sll $t3, $t2, 2

add $t3, $t6, $t3 # Calculate the address of A[i][k]

li $t4, 12

mult $t4, $t2 # Calculate the offset for B[k][j]

mflo $t4

add $t5, $t8, $t4 # Calculate the address of B[k][j]

lw $t3, 0($t3) # Load A[i][k] into $t3

lw $t5, 0($t5) # Load B[k][j] into $t5

mul $t3, $t3, $t5 # Multiply A[i][k] and B[k][j]

lw $t5, 0($t9) # Load the current value of C[i][j]

add $t3, $t3, $t5 # Add the product to C[i][j]

sw $t3, 0($t9) # Store the result in C[i][j]

addi $t2, $t2, 1 # Increment k

j loop\_k # Jump to beginning of innermost loop

next\_k:

addi $t1, $t1, 1 # Increment j

j loop\_j # Jump to beginning of inner loop

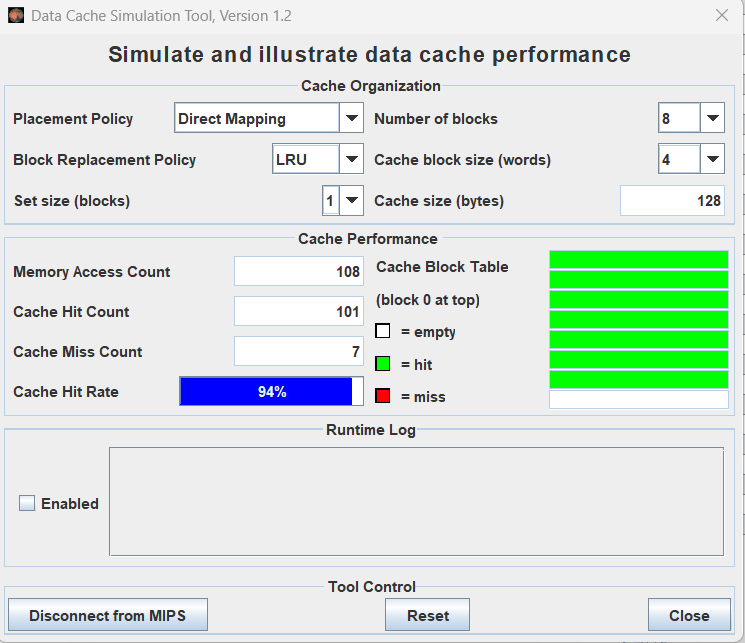
next\_j:

addi $t0, $t0, 1 # Increment i

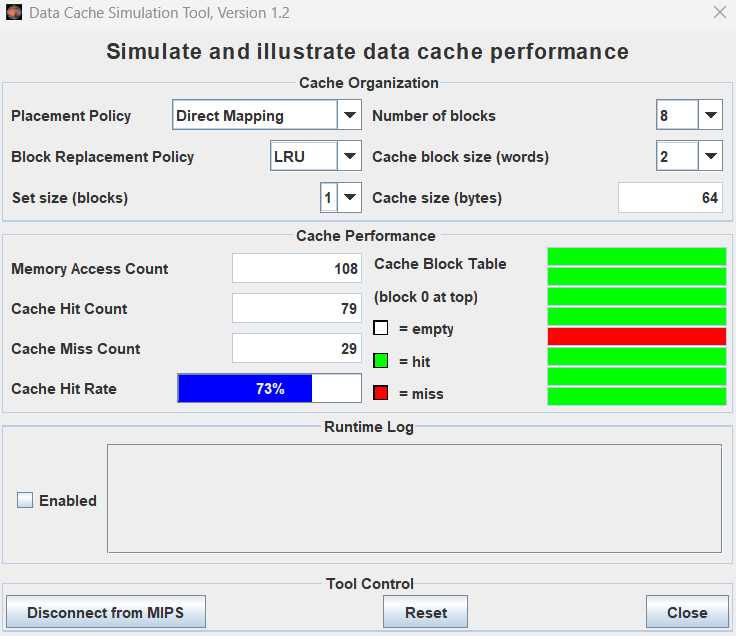
j loop\_i # Jump to beginning of outer loop

exit:

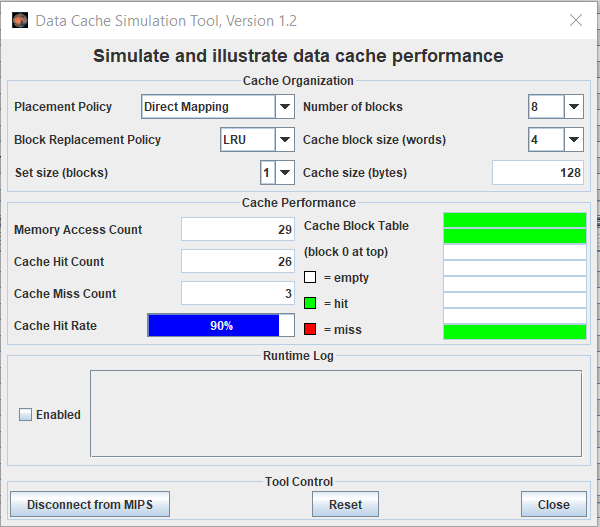
**For matrix multiplication: -**

****

**On reducing block size**

****

**For Fibonacci :**

****

**Your Analysis Here:**

* The code of Fibonacci numbers has higher hit rate than the matrix multiplication one when both are compared on same test benches.
* While decreasing the block size the hit rate decreases. And while increasing the no. of blocks the hit rate increases .

**Submission:**

**Demonstrate to TAs your results and**

Submit code and Report through

<https://u.pcloud.com/#page=puplink&code=m6wkZiR1xLpmF9TVKci3TqvWYvX4T8oBk>