**CS210: (Lab 15)**

**Computer Architecture Lab**

**Cache Performance Analysis and Branch Prediction**

**Task 1:** 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]];

}

Ans:

main:

addiu $sp,$sp,0x10010004

sw $fp,0($sp)

move $fp,$sp

sw $0,0($fp)

b .L2

nop

.L3:

lw $2,0($fp)

sll $2,$2,2

addu $2,$fp,$2

lw $2,4($2)

sll $2,$2,2

addu $2,$fp,$2

lw $3,4($2)

lw $2,0($fp)

sll $2,$2,2

addu $2,$fp,$2

sw $3,404($2)

lw $2,0($fp)

addiu $2,$2,1

sw $2,0($fp)

.L2:

lw $2,0($fp)

addi $t2,$t2,100

slt $2,$2,$t2

bne $2,$0,.L3

nop

move $2,$0

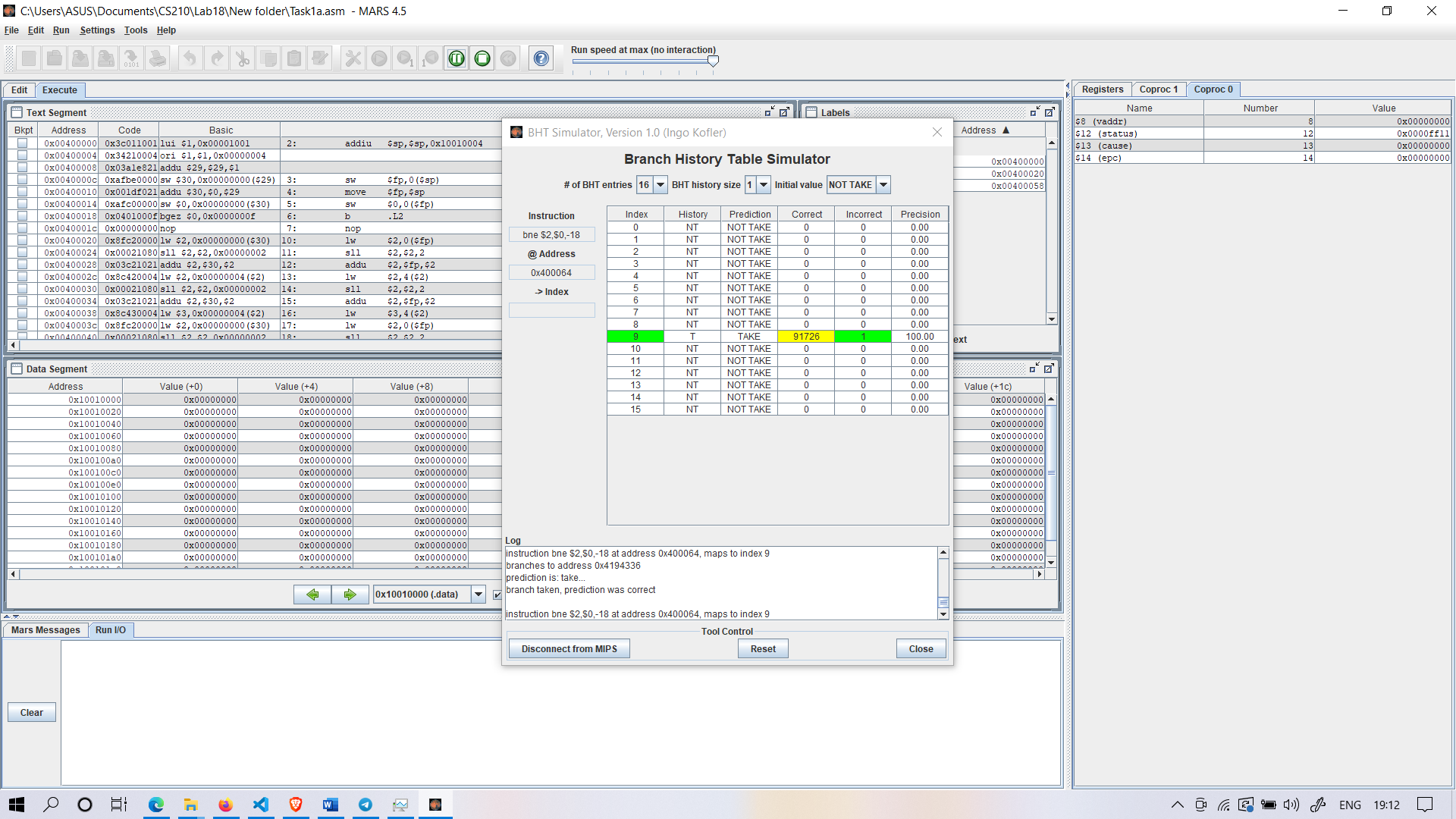
move $sp,$fp

lw $fp,812($sp)

addiu $sp,$sp,816

jr $31

nop

****

BHT Simulator for 1(a)

Analysis : It is observed that that on Selecting NOT TAKEN, Only 1 address is missed

So Hit rate ~ 100%

**(b)**

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

A[i] = i;

addi $t0, $0, 0

addi $t1, $0, 100

addi $sp, $0, 0x10010004

while: beq $t0, $t1, end

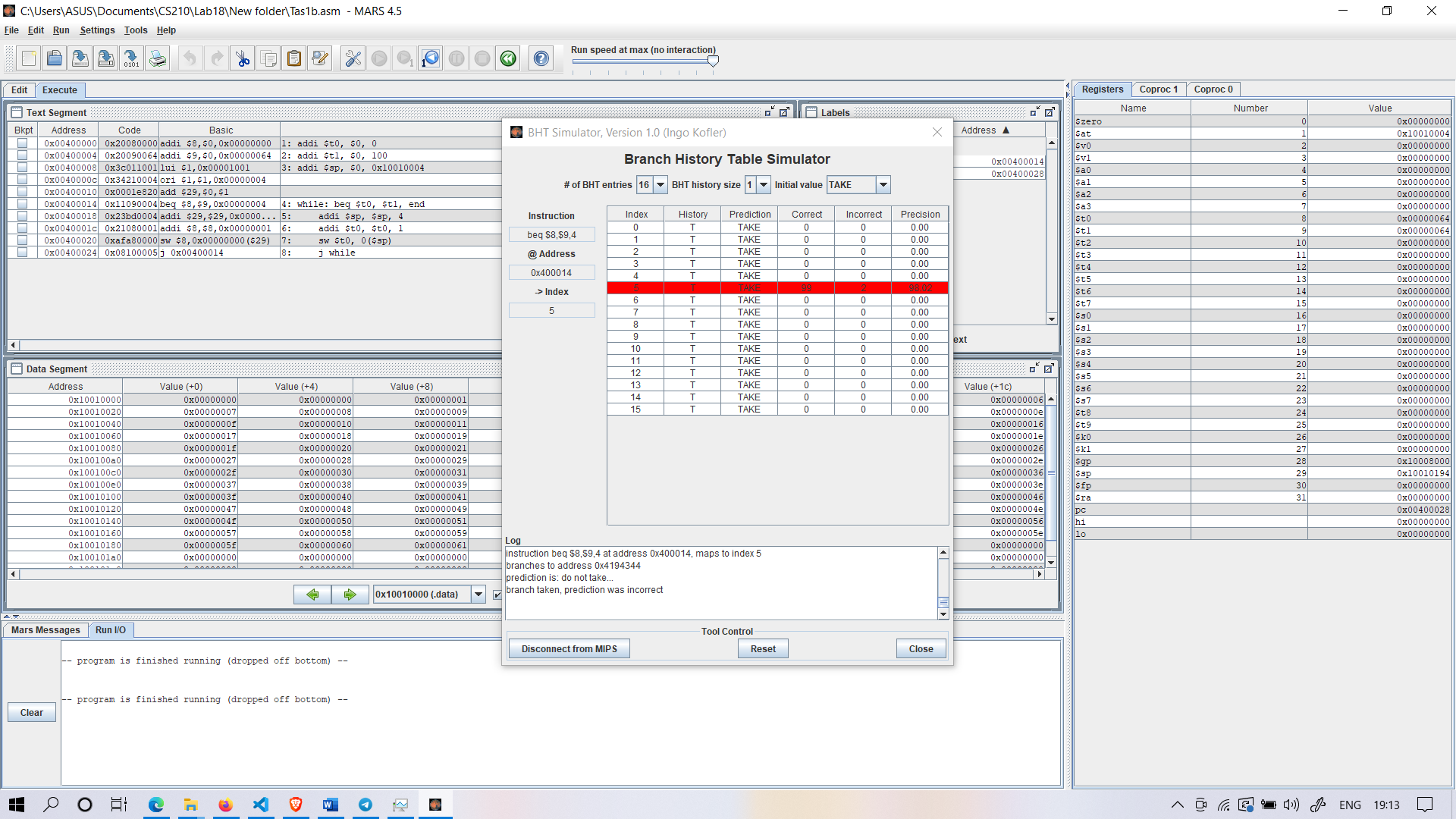
addi $sp, $sp, 4

addi $t0, $t0, 1

sw $t0, 0($sp)

j while

end:

****

BHT Simulator for 1(b)

Analysis : It is observed that that on Selecting TAKEN, Only 2 address is missed

So Hit rate ~ 99%

**Task 2:** Open the program row-major.asm from the Examples folder if it is not already open. Assemble the program. From the Tools menu, select Memory Reference Visualization. Visualize memory content for this.

**Ans:**

.data

data: .word 0 : 256 # storage for 16x16 matrix of words

.text

li $t0, 16 # $t0 = number of rows

li $t1, 16 # $t1 = number of columns

move $s0, $zero # $s0 = row counter

move $s1, $zero # $s1 = column counter

move $t2, $zero # $t2 = the value to be stored

loop: mult $s0, $t1 # $s2 = row \* #cols (two-instruction sequence)

mflo $s2 # move multiply result from lo register to $s2

add $s2, $s2, $s1 # $s2 += column counter

sll $s2, $s2, 2 # $s2 \*= 4 (shift left 2 bits) for byte offset

sw $t2, data($s2) # store the value in matrix element

addi $t2, $t2, 1 # increment value to be stored

addi $s1, $s1, 1 # increment column counter

bne $s1, $t1, loop # not at end of row so loop back

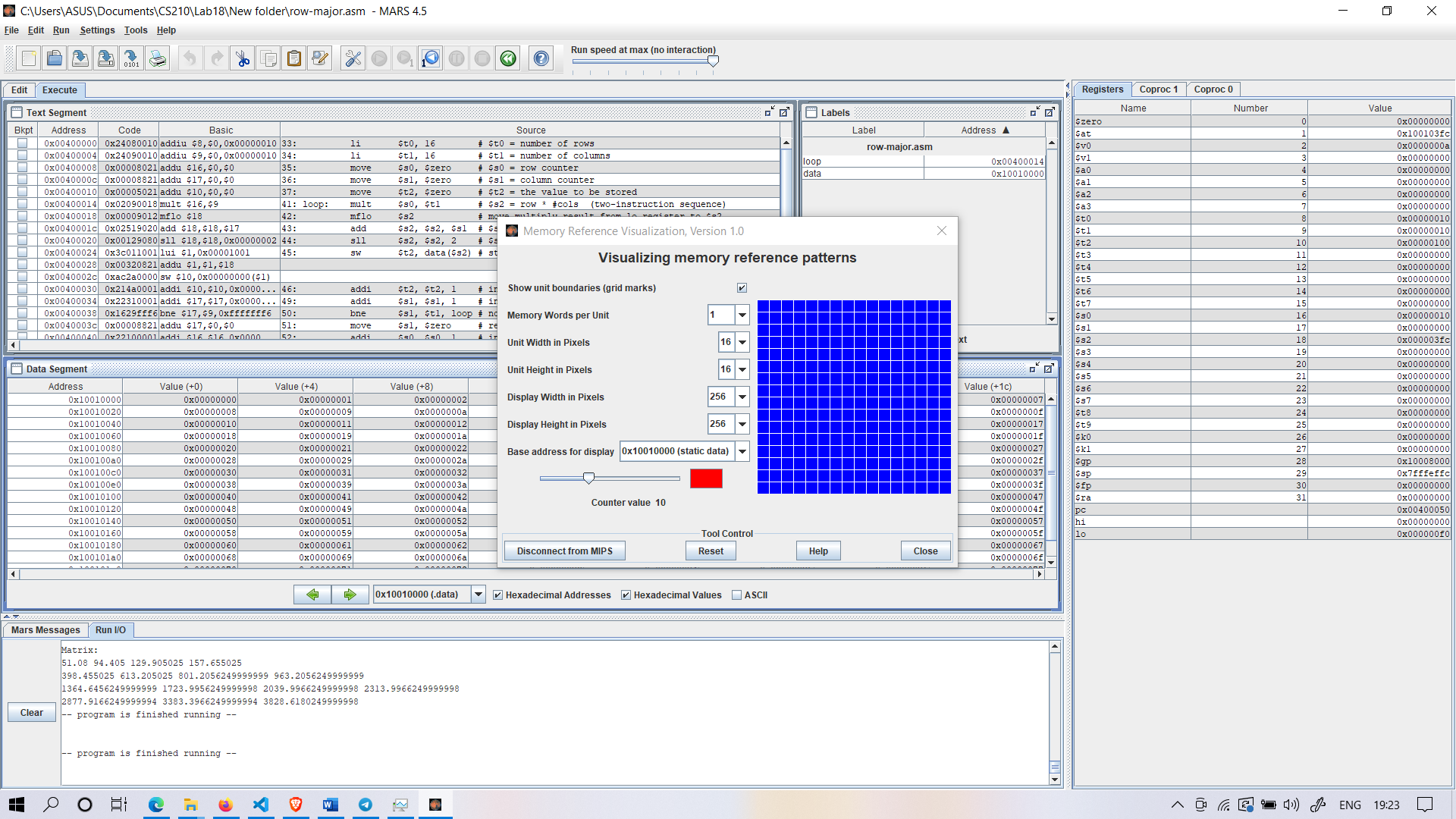
move $s1, $zero # reset column counter

addi $s0, $s0, 1 # increment row counter

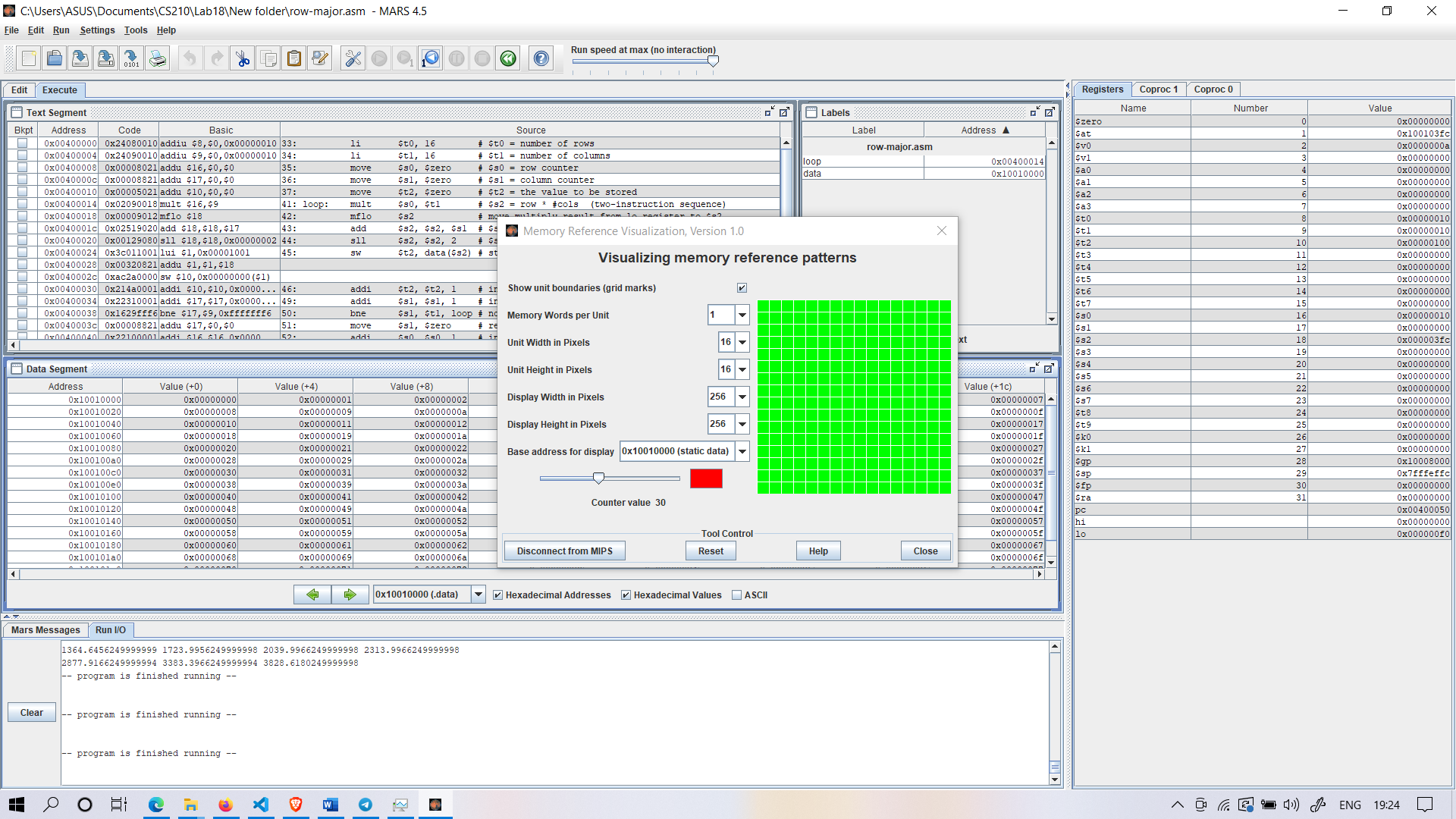
bne $s0, $t0, loop # not at end of matrix so loop back

li $v0, 10 # system service 10 is exit

syscall # we are outta here.

****

**Memory Reference At Counter Value 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:** Create two examples algorithms which demonstrate efficient use of data cache. Conduct performance analysis on different parameters such as block size, cache replacement algorithms.

**Ans:**

**1.Matrix Multiplication:**

.data

.align 3

newMatrix: .space 96 #space for 3x4 array

.align 3

firstMatrix: .double 0.50, 0.25, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.15, 16.16

.align 3

secondMatrix: .double 16.16, 15.15, 14.0, 13.0, 12.0, 11.0, 10.0001, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0

newLine\_prompt: .asciiz "\n"

space\_prompt: .asciiz " "

print\_array\_prompt: .asciiz "Matrix:"

.text

.globl main

main:

addi $sp, $sp, -16

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

sw $s2, 12($sp)

la $a0, firstMatrix

jal print\_matrix

jal print\_new\_line

la $a0, secondMatrix

jal print\_matrix

la $a0, firstMatrix

la $a1, secondMatrix

la $a2, newMatrix

jal matrix\_mult

jal print\_new\_line

jal print\_new\_line

la $a0, newMatrix

jal print\_matrix

li $v0, 10

syscall

lw $s2, 12($sp)

lw $s1, 8($sp)

lw $s0, 4($sp)

lw $ra, 0($sp)

addi $sp, $sp, 16

jr $ra

matrix\_mult:

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)

# f4 = product(f0, f1)

# f6 = newMatrix[i][j]

or $s0, $s0, $a0

or $s1, $s1, $a1

or $s2, $s2, $a2

ori $s3, $s3, 0 # s0 = i = 0

ori $s4, $s4, 0 # s1 = j = 0

ori $s5, $s5, 0 # s2 = k = 0

ori $s6, $s6, 4 # s3 = n = 3

matrix\_mult\_loop1:

li $s4, 0

matrix\_mult\_loop2:

li $s5, 0

li $t6, 0

matrix\_mult\_loop3:

#assign f0 to firstMatrix[i][k]

# t0 = i \* n

# t1 = t0 + i

mult $s3, $s6

mflo $t0

add $t1, $t0, $s5

sll $t1, $t1, 3

add $t2, $t1,$s0

l.d $f0, 0($t2) # f0 = firstMatrix[i][k]

#assign f2 to firstMatrix[k][i]

# t0 = k \* n

# t1 = t0 + j

mult $s5, $s6

mflo $t0

add $t1, $t0, $s4

sll $t1, $t1, 3

add $t2, $t1, $s1

l.d $f2, 0($t2) # f2 = secondMatrix[k][j]

mul.d $f4, $f0, $f2

add.d $f6, $f6, $f4

addi $s5, $s5, 1 # s2 = ++k

bne $s5, $s6, matrix\_mult\_loop3 # if(k != 4)

# if(k == 4)

# t0 = i \* n

# t1 = t0 + j

mult $s3, $s6 # store z[i][j]

mflo $t0

add $t1, $t0, $s4 #i\*n+j

sll $t1, $t1, 3 #8\*(i\*n+j)

add $t2, $t1, $s2 #&z[i][j]

s.d $f6, 0($t2)

addi $s4, $s4, 1 # s1 = ++j

bne $s4, $s6, matrix\_mult\_loop2 # if(j != 4)

# if(j == 4)

addi $s3, $s3, 1 # s0 = ++i

bne $s3, $s6, matrix\_mult\_loop1 # if(i != 4)

# if(i == 4)

#End of matrixMult method

lw $s6, 28($sp)

lw $s5, 24($sp)

lw $s4, 20($sp)

lw $s3, 16($sp)

lw $s2, 12($sp)

lw $s1, 8($sp)

lw $s0, 4($sp)

lw $ra, 0($sp)

addi $sp, $sp, 32

jr $ra

print\_matrix:

addi $sp, $sp, -12

sw $ra, 0($sp)

sw $s0, 4($sp) # s0 = current array

sw $s1, 8($sp) # s1 = i

sw $s2, 12($sp) # s2 = n

# a0 = current array

or $s0, $a0, $zero

# print array prompt

la $a0, print\_array\_prompt

li $v0, 4

syscall

# print new line

jal print\_new\_line

li $s1, 0 # s1 = i = 0

li $s2, 15 # s2 = n = 15

print\_matrix\_loop:

sll $t0, $s1, 3

add $t1, $t0, $s0

l.d $f12, 0($t1)

# print out current item

li $v0, 3

syscall

# print out a space

jal print\_space

# if(i % 4 == 0)

li $t0, 4

addi $s1, $s1, 1

div $s1, $t0

mfhi $t0

bne $t0, $zero, skip\_new\_line

jal print\_new\_line

skip\_new\_line: # else if(i % 4 != 0)

bne $s1, $s2, print\_matrix\_loop # if(i != n)

lw $s2, 12($sp)

lw $s1, 8($sp)

lw $s0, 4($sp)

lw $ra, 0($sp)

addi $sp, $sp, 12

jr $ra

print\_new\_line:

addi $sp, $sp, -4

sw $ra, 0($sp)

la $a0, newLine\_prompt

li $v0, 4

syscall

lw $ra, 0($sp)

addi $sp, $sp, 4

jr $ra

print\_space:

addi $sp, $sp, -4

sw $ra, 0($sp)

la $a0, space\_prompt

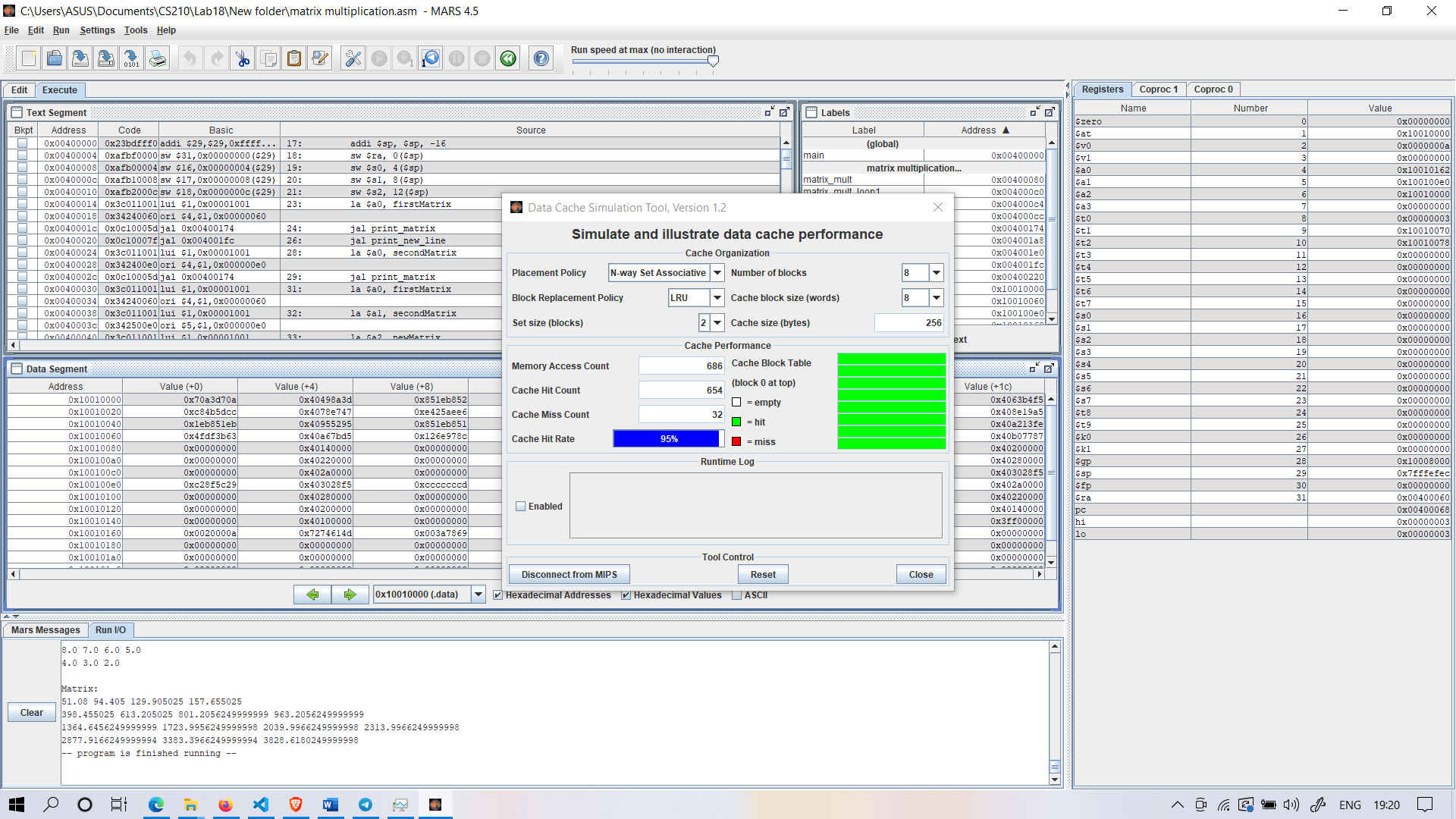
li $v0, 4

syscall

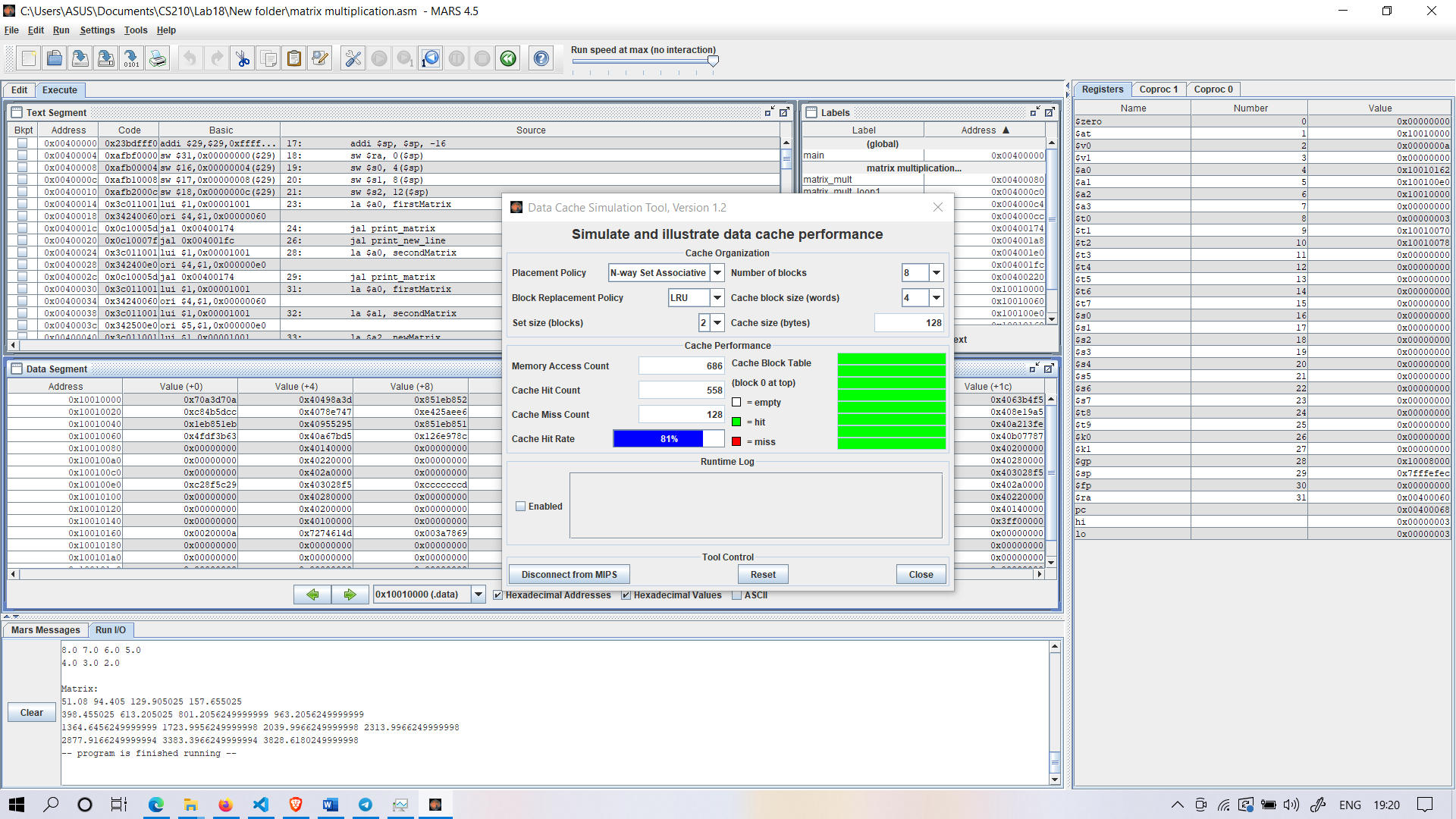
lw $ra, 0($sp)

addi $sp, $sp, 4

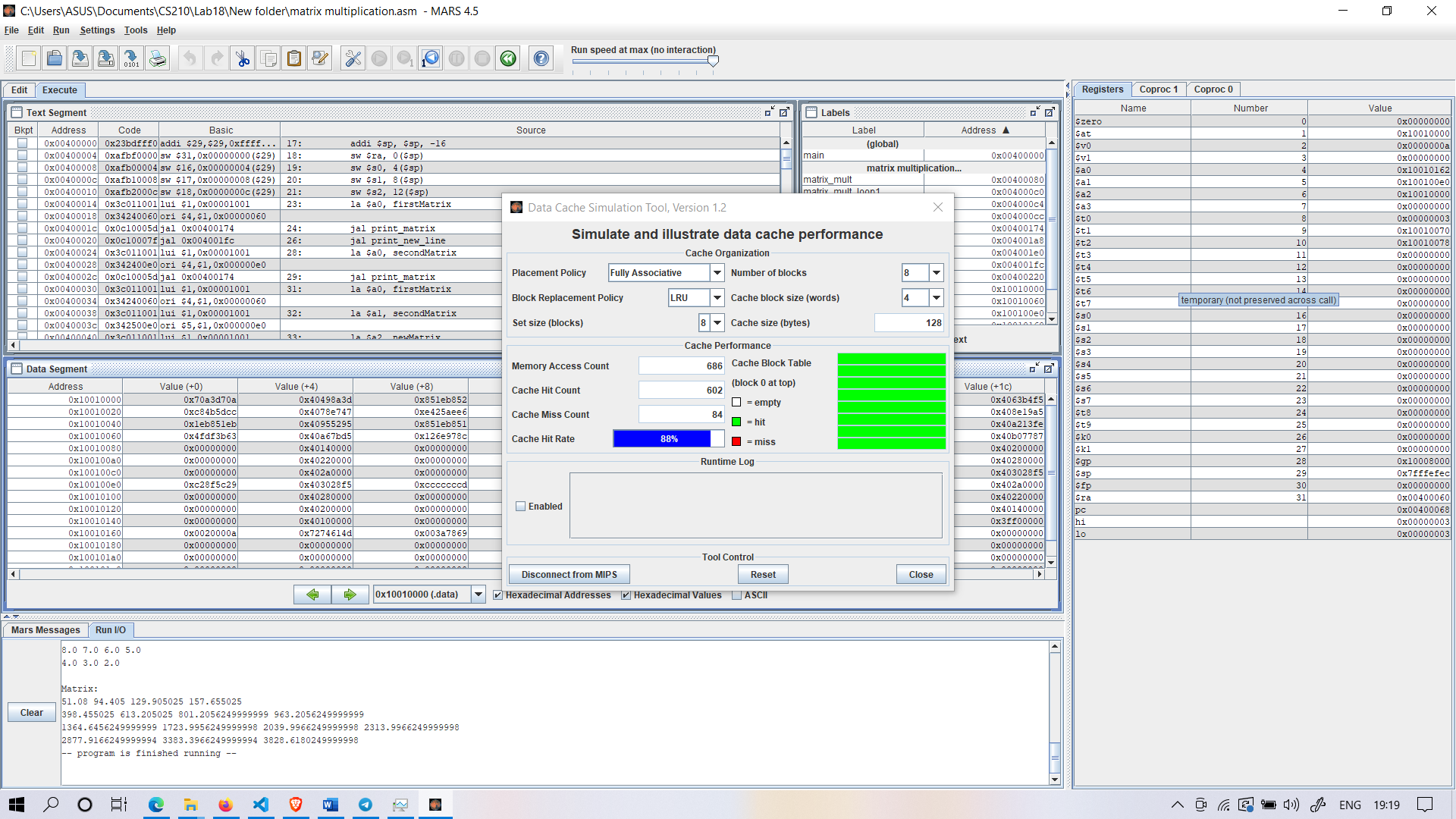
jr $ra

****

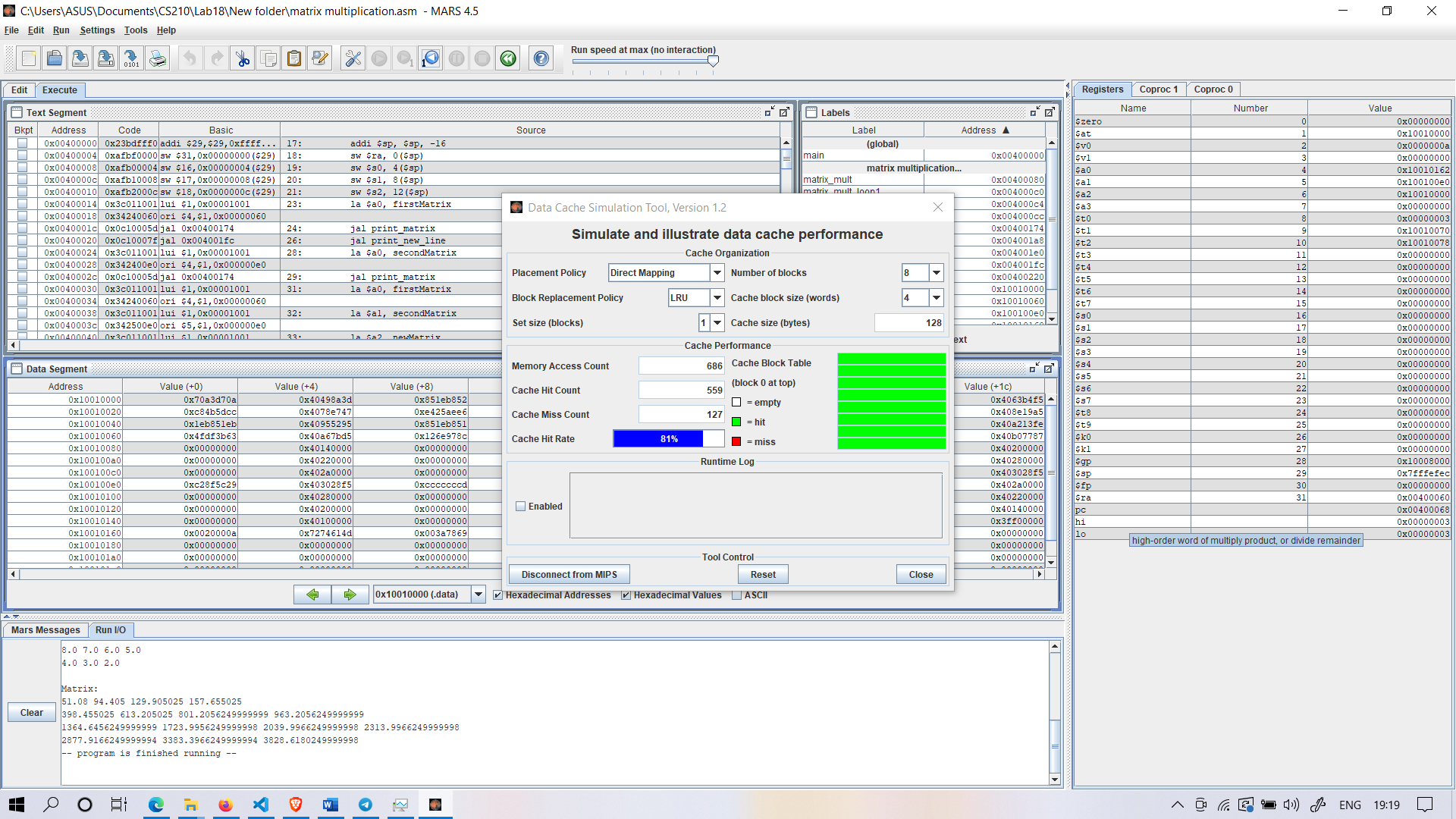
N-Set Associative,Block Size=8, Hit Rate=95%

****

N-Set Associative,Block Size=4, Hit Rate=81%

****

Fully Associative,Block Size=4, Hit Rate=88%

****

Direct Mapping,Block Size=4, Hit Rate=81%

**2. Fibonacci Numbers**

.data

prompt1: .asciiz "PLEASE ENTER THE VALUE OF INDEX : \n"

prompt2: .asciiz "FIBONACII NUMBER AT iTH INDEX :\n"

.text

# PRINT PROMPT1

li $v0, 4

la $a0, prompt1

syscall

# READ STRING

li $v0, 5

syscall

# CALL FIBONACCI

move $a0, $v0

jal fibonacci

move $a1, $v0 # SAVE RETURN VALUE TO A1

# PRINT PROMPT2

li $v0, 4

la $a0, prompt2

syscall

# PRINT RESULT

li $v0, 1

move $a0, $a1

syscall

# EXIT

li $v0, 10

syscall

fibonacci:

addi $sp, $sp, -12

sw $ra, 8($sp)

sw $s0, 4($sp)

sw $s1, 0($sp)

move $s0, $a0

li $v0, 1 # RETURN VALUE FOR TERMINAL CONDITION

ble $s0, 0x2, fibonacciExit # CHECK TERMINAL CONDITION

addi $a0, $s0, -1 # SET ARGS FOR RECURSIVE CALL TO F(N-1)

jal fibonacci

move $s1, $v0 # STORE RESULT OF F(N-1) TO S1

addi $a0, $s0, -2 # SET ARGS FOR RECURSIVE CALL TO F(N-2)

jal fibonacci

add $v0, $s1, $v0 # ADD RESULT OF F(N-1) TO IT

fibonacciExit:

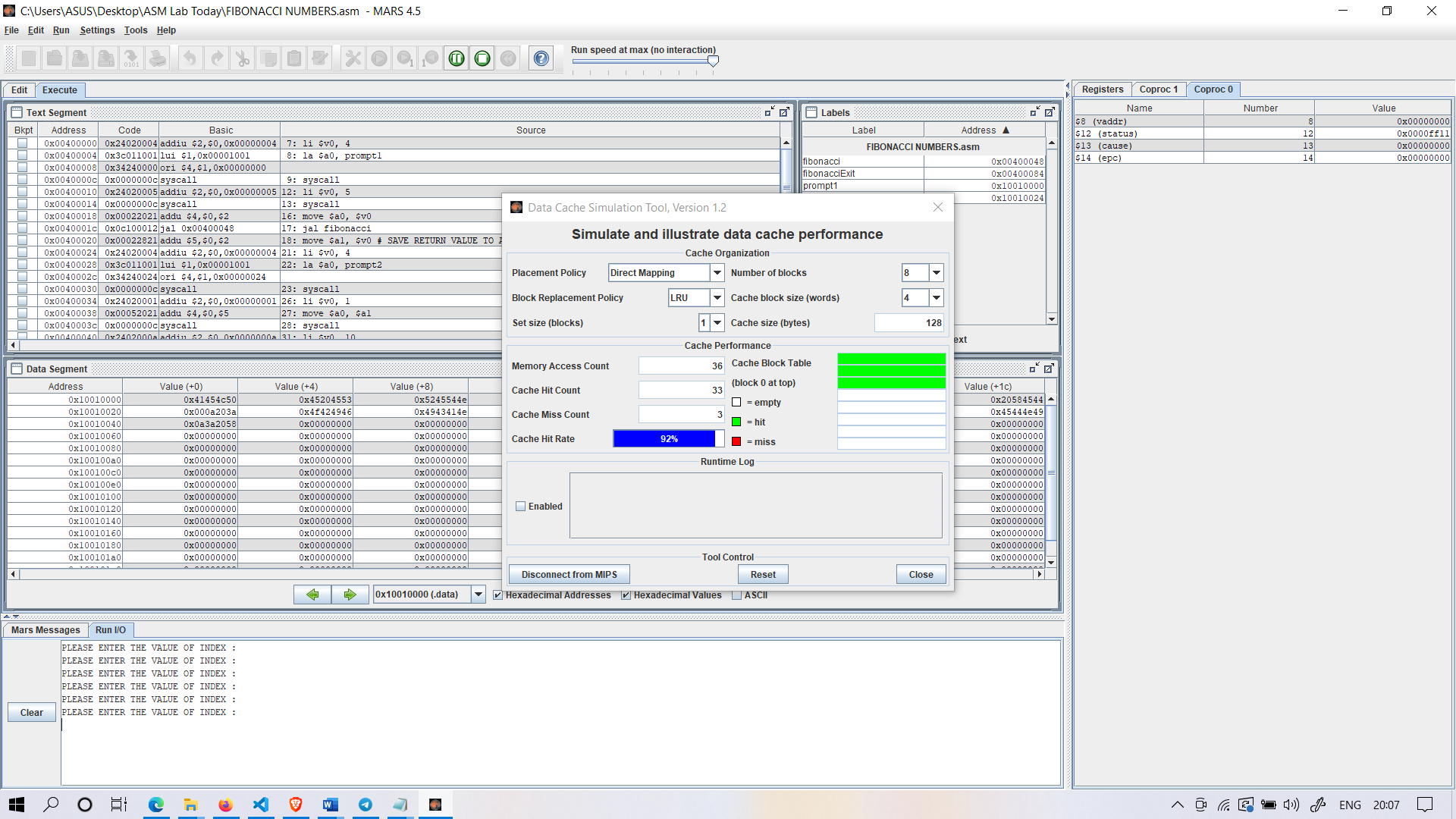
lw $ra, 8($sp)

lw $s0, 4($sp)

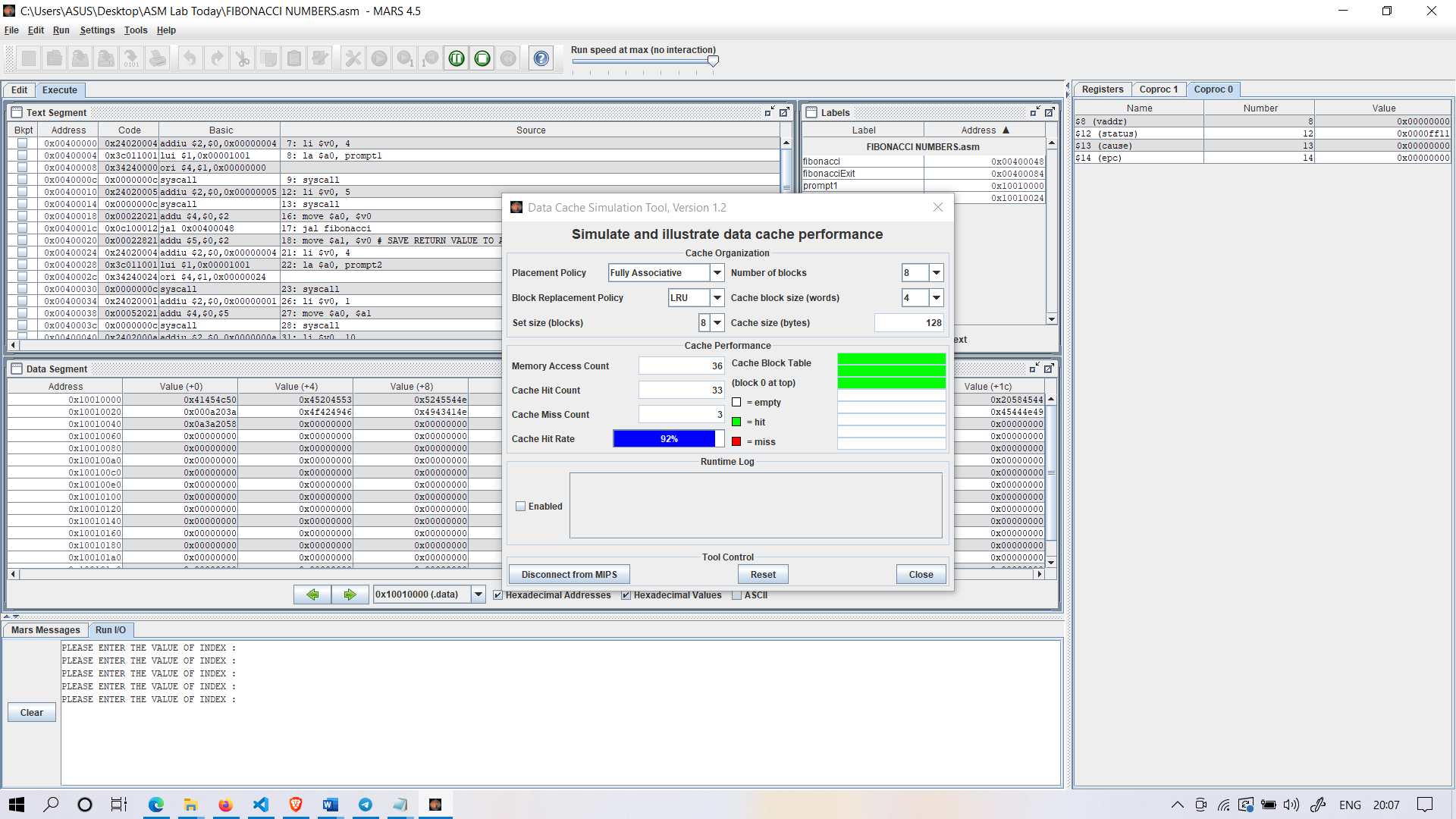
lw $s1, 0($sp)

addi $sp, $sp, 12

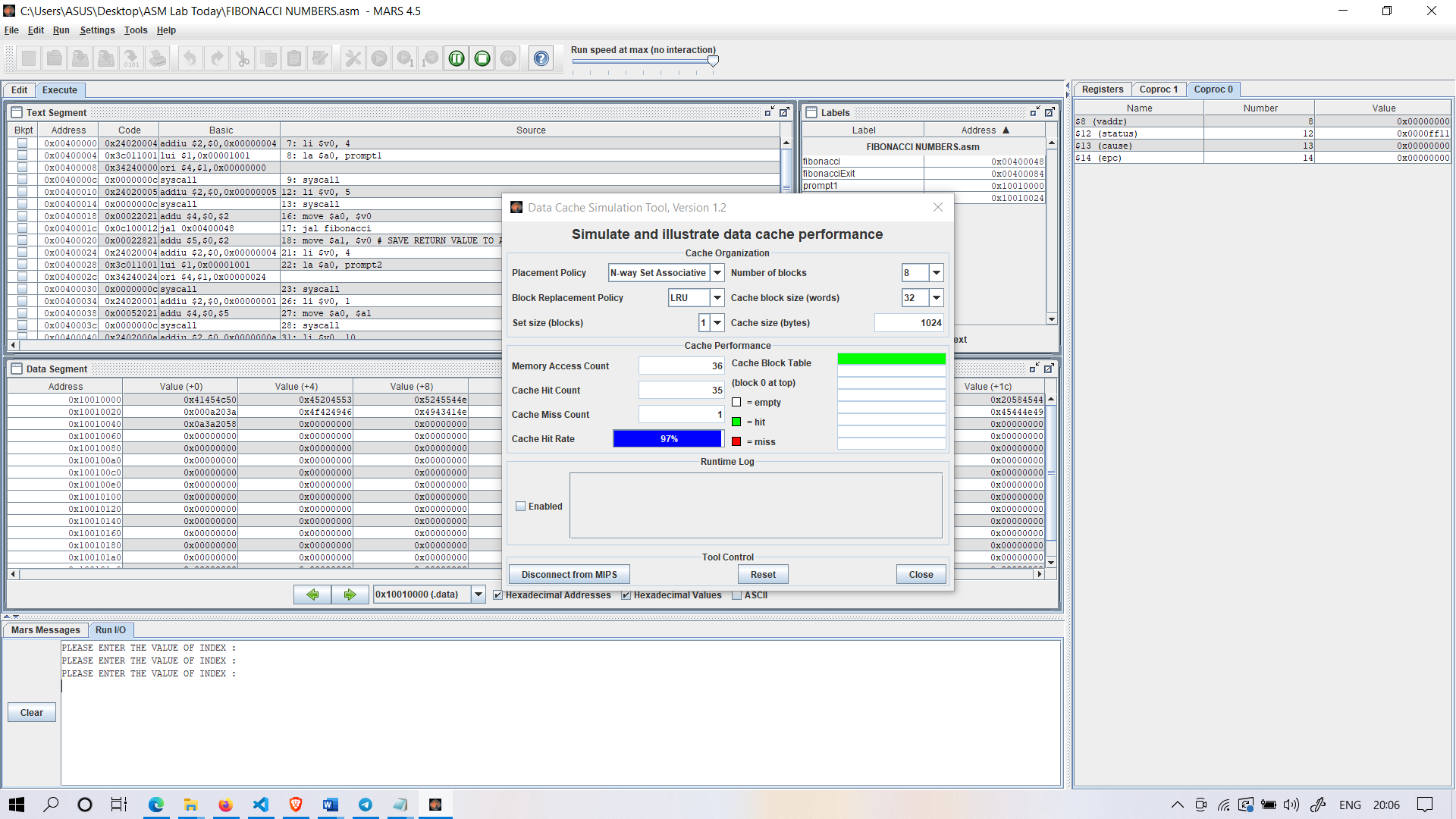
jr $ra



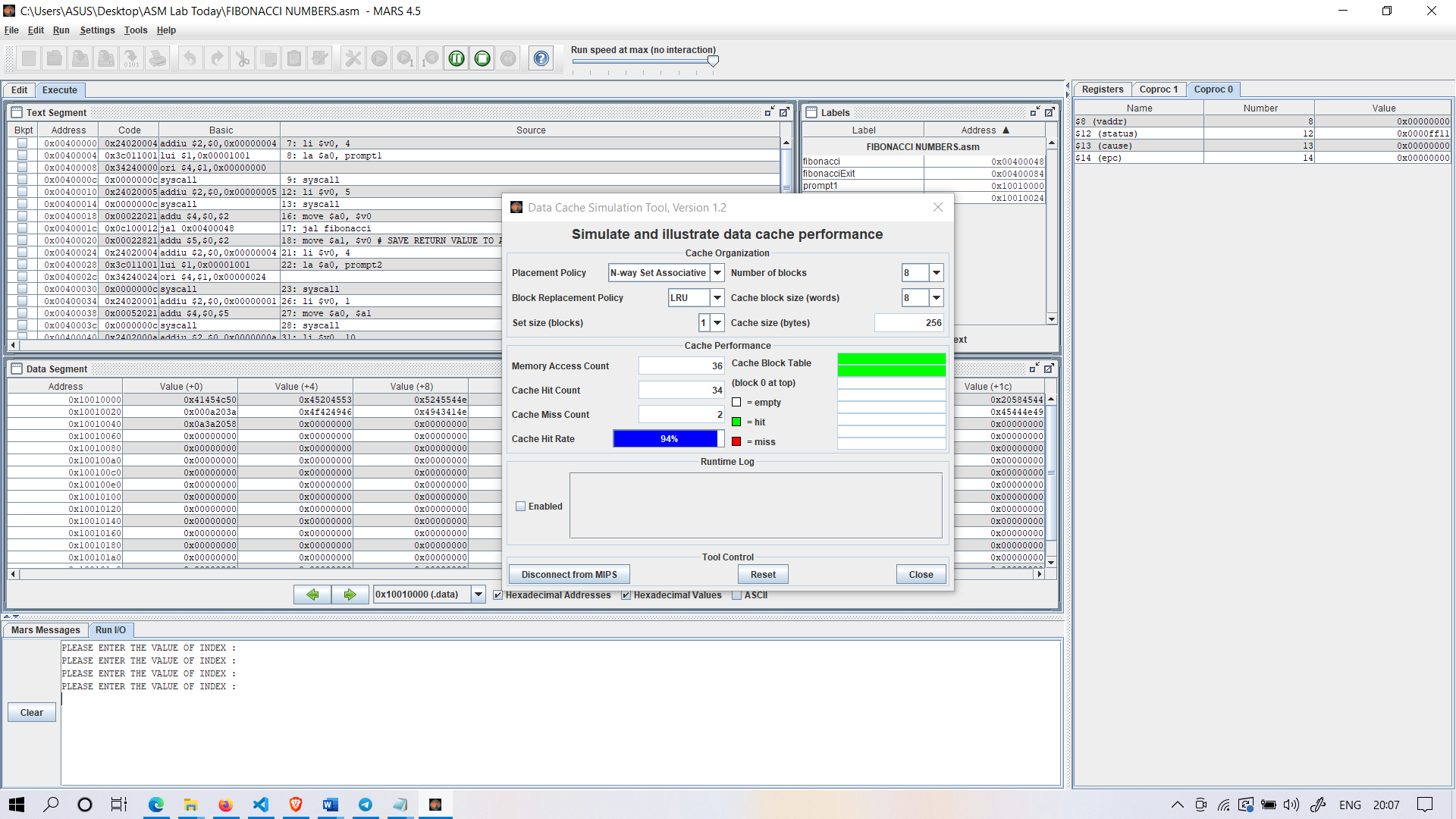
Direct Mapping, Block Size=4, Hit Rate=92%



Fully Set Associative,Block Size=4, Hit Rate=92%



N-Set Associative,Block Size=32, Hit Rate=97%



N-Set Associative,Block Size=8, Hit Rate=94%

Analysis :

* Increasing Block Size increases Hit rate
* Increasing Associativity increases Hit rate
* Increasing Cache Size increases Hit rate