**Computer Architecture 3/System Software 2**

**Project Mark Sheet**

|  |  |
| --- | --- |
| **TEAM MEMBERS**   * **Student Number, Surname and Initials** | 219290148 Maredi PP |
| 220645264 Makeleni Y |
| 219449236 Makasi L |
| 219359253 Makalima C |
| 219493154 Lwanga AMK |
|  |
| **NAME OF TEAM** | PYLAC |
| **SELECTED PROJECT** | Matrix Multiplication |
| **TEAM LEADER**   * **Student Number, Surname and Initials** | 219290148 Maredi PP |

1. Insert code

|  |  |
| --- | --- |
| **Part I**  **Append first version of code**  **[35]** | (Pasted Below) |

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

# REGISTERS RELATION #

# $t0 - MULTI-COUNTER/ COUNTER FOR LOOP K #

# $t1 - ADDRESS OF DIM/ADDRESS OF MATBT #

# $t2 - ADDRESS OF MATA #

# $t3 - SCANNED NUMBER/ADDRESS OF SPACE #

# $t4 - DIMENSION #

# $t5 - A[i]/DOUBLED DIMENSION #

# $t6 - BT[i]/ADDRESS OF MESSAGE 2 #

# $t7 - MULT = A[i] \* B[i] #

# $s0 - ADDRESS OF MESG1/ADDRESS OF TWOPTS/COUNTER FOR#

# LOOP J; COMPARISON PURPOSES #

# $s1 - TEMPORAL COUNTER #

# $s2 - COUNTER FOR LOOPI #

# $s3 - END OF LINE #

# $s4 - (C[i]) SUM = SUM + MULT #

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

.data

prompt: .asciiz "The Dimension of The Matrices Will Be: 2 x 2.\nNow, You Must Define The Numbers for The Matrix A (4 numbers), Then Immediately The Numbers for The Matrix B Will Be Asked: \n"

twopts: .asciiz ": "

newLine: .asciiz "\n"

space : .asciiz " "

matrixA: .asciiz "Contents of Matrix A"

matrixB: .asciiz "Contents of Matrix B"

matrixC: .asciiz "Contents of Matrix C"

.align 4

matrixElements: .space 32

MATA : .byte 00

.byte 00

.byte 00

.byte 00

MATB: .byte 00

.byte 00

.byte 00

.byte 00

MATC: .byte 00

.byte 00

.byte 00

.byte 00

MATBT: .byte 00

.byte 00

.byte 00

.byte 00

DIM: .byte 4

.text

main:

xor $t0, $t0, $t0 # Counter Initialization

la $t1, DIM # Load the address of DIM (Dimension) into $t1

la $t2, MATA # Load the address of MATA (Matrix A) into $t2

la $s0, prompt # Load the address of Prompt Message into $s0

la $s3, newLine # Load the address of the End of Line into $s3

lbu $t4, 0($t1) # Load the value of DIM

add $t5, $t4, $t4 # Temporal Variable for the condition in "ragain"

addi $t9, $0, 0

li $v0, 4 # Print Prompt Message

la $a0, 0($s0)

syscall

la $s0, twopts # Load the address of 2Points into $s0

read:

li $v0, 1 # Print the index of the matrix

move $a0, $t0

syscall

li $v0, 4 # Print 2Points

la $a0, 0($s0)

syscall

li $v0, 5 # Read the Integer

syscall

add $t3, $0, $v0 # Store the number given by the user into $t3

sw $v0, matrixElements($t9)

addi $t9, $t9, 4

sb $t3, 0($t2) # Store the number in the Matrix

addi $t2, $t2, 1 # Increase by 1 the Index, the Counter and the Temporal Counter ($s1)

addi $t0, $t0, 1

addi $s1, $s1, 1

bne $t4, $t0, read # If the counter it is not equal to the dimension, keep reading

ragain:

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

xor $t0, $t0, $t0 # Reinitialize Counter

bne $s1, $t5, read # Read Again; Now for Matrix B

intlz1:

la $t2, MATA # Reload the address of MATA (Matrix A) into $t2

la $t3, space # Load the address of Spaces into $t3

xor $s1, $s1, $s1 # Reinitialize Temporal Counter

la $t1, MATBT # Load the address of MATBT into $t1

trnpse:

lbu $t6, 4($t2) # Load B[i]

sb $t6, 0($t1) # Store B[i] into it's transpose (BT[i])

addi $t2, $t2, 2 # Offset for the Transpose

addi $t0, $t0, 2 # Increase counter in 2

addi $t1, $t1, 1 # Increase by 1 the index of BT[i]

bne $t0, $t4, trnpse # If the 2 numbers of a line are printed, go to the next one

addi $s1, $s1, 1 # Next Line

la $t2, MATA # Reload the address of MATA into $t2

xor $t0, $t0, $t0 # Reinitialize Counter

add $t2, $t2, $s1 # Increase the offset for $t2

bne $s1, 2, trnpse # If the 2 lines are not printed, repeat the process

#>>>>>>>>>>>>>>>>>>>>Printing Elements of Matrix A<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

printMatrixA:

addi $t0, $zero, 0

li $v0, 4 # Print Matrix A Message

la $a0, matrixA

syscall

matrixANewLine:

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

addi $t9, $zero, 0

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

printMatrixAElements:

beq $t0, 16, printMatrixB

lw $t8, matrixElements($t0)

addi $t0, $t0, 4

li $v0, 1

move $a0, $t8

syscall

li $v0, 4 # Print the spaces between numbers

la $a0, space

syscall

addi $t9, $t9, 1

beq $t9, 2, matrixANewLine # If all the numbers of the Matrix A haven't been printed, do all the same process

j printMatrixAElements

#>>>>>>>>>>>>>>>>>>>>Done Printing Elements of Matrix A<<<<<<<<<<<<<<<<<<<<<<<<<<<<

#>>>>>>>>>>>>>>>>>>>>Printing Elements of Matrix B<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

printMatrixB:

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

addi $t0, $zero, 16

li $v0, 4 # Print Matrix B Message

la $a0, matrixB

syscall

matrixBNewLine:

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

addi $t9, $zero, 0

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

printMatrixBElements:

beq $t0, 32, intlz2

lw $t8, matrixElements($t0)

addi $t0, $t0, 4

li $v0, 1

move $a0, $t8

syscall

li $v0, 4 # Print the spaces between numbers

la $a0, space

syscall

addi $t9, $t9, 1

beq $t9, 2, matrixBNewLine # If all the numbers of the Matrix B haven't been printed, do all the same process

j printMatrixBElements

#>>>>>>>>>>>>>>>>>>>>Done Printing Elements of Matrix B<<<<<<<<<<<<<<<<<<<<<<<<<<<<

intlz2:

li $v0, 4 # Call service in order to print an "End of Line" twice

la $a0, 0($s3)

syscall

la $t6, matrixC # Load the address of Message (Matrix C) into $t6

li $s2, -2 # MATA Offset Counter; Counter for Loop i

xor $s1, $s1, $s1 # Reinitialize Temporal Counter

li $v0, 4 # Print Message 2

la $a0, 0($t6)

syscall

loopi:

xor $s0, $s0, $s0 # Initialize counter for loop j

la $t1, MATBT # Reload the address of MATBT into $t1

addi $s2, $s2, 2 # Increase in 2 the counter for loop i

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

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

loopj:

beq $s0, $t4, loopi # If $s0 it's equal to $t4, go to loop i

la $t2, MATA # Reload the address of MATA into $t2

add $t2, $t2, $s2 # Increase the index of A[i] depending the current state of the process

xor $s4, $s4, $s4 # Reinitialize Sum for C[i]

xor $t0, $t0, $t0 # Reinitialize Counter for loop k

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

loopk:

lbu $t5, 0($t2) # Load A[i]

lbu $t6, 0($t1) # Load BT[i]

mul $t7, $t5, $t6 # $t7 = A[i] \* B[i]

add $s4, $s4, $t7 # sum = sum + $t7

addi $t2, $t2, 1 # Increase by 1 the indices of A[i] & BT[i];

addi $t1, $t1, 1

addi $t0, $t0, 1 # Increase by 1 the counter for loop k & the counter for loop j

addi $s0, $s0, 1

bne $t0, 2, loopk # If $t0 it's not equals to 2, go to loop k

addi $s1, $s1, 1 # Increase by 1 the counter of the index of MATC

la $t2, MATA # Reload the address of MATA into $t2

add $t2, $t2, $s1 # Increase the index of C[i]

sb $s4, 17($t2) # Store the value of $s4 into MATC

li $v0, 1 # Print number of C[i]

add $a0, $0, $s4

syscall

li $v0, 4 # Print the spaces between numbers

la $a0, 0($t3)

syscall

bne $s1, $t4, loopj # If all the numbers of the MATC haven't been printed, do all the same process

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

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

exit:

li $v0, 10 # Exit Program

syscall

|  |  |
| --- | --- |
| **Part II**  **Append second version of code**  **[25]** | (Pasted Below) |

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

# REGISTERS RELATION #

# $t0 - MULTI-COUNTER/ COUNTER FOR LOOP K #

# $t1 - ADDRESS OF DIM/ADDRESS OF MATBT #

# $t2 - ADDRESS OF MATA #

# $t3 - SCANNED NUMBER/ADDRESS OF SPACE #

# $t4 - DIMENSION #

# $t5 - A[i]/DOUBLED DIMENSION #

# $t6 - BT[i]/ADDRESS OF MESSAGE 2 #

# $t7 - MULT = A[i] \* B[i] #

# $s0 - ADDRESS OF MESG1/ADDRESS OF TWOPTS/COUNTER FOR#

# LOOP J; COMPARISON PURPOSES #

# $s1 - TEMPORAL COUNTER #

# $s2 - COUNTER FOR LOOPI #

# $s3 - END OF LINE #

# $s4 - (C[i]) SUM = SUM + MULT #

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

.data

prompt: .asciiz "The Dimension of The Matrices Will Be: 3 x 3.\nNow, You Must Define The Numbers for The Matrix A (9 numbers), Then Immediately The Numbers for The Matrix B Will Be Asked: \n"

twopts: .asciiz ": "

newLine: .asciiz "\n"

space : .asciiz " "

matrixA: .asciiz "Contents of Matrix A"

matrixB: .asciiz "Contents of Matrix B"

matrixC: .asciiz "Contents of Matrix C"

.align 4

matrixElements: .space 72

MATA : .byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

MATB: .byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

MATC: .byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

MATBT: .byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

.byte 00

DIM: .byte 9

.text

main:

xor $t0, $t0, $t0 # Counter Initialization

la $t1, DIM # Load the address of DIM (Dimension) into $t1

la $t2, MATA # Load the address of MATA (Matrix A) into $t2

la $s0, prompt # Load the address of Prompt Message into $s0

la $s3, newLine # Load the address of the End of Line into $s3

lbu $t4, 0($t1) # Load the value of DIM

add $t5, $t4, $t4 # Temporal Variable for the condition in "ragain"

addi $t9, $0, 0

li $v0, 4 # Print Prompt Message

la $a0, 0($s0)

syscall

la $s0, twopts # Load the address of 2Points into $s0

read:

li $v0, 1 # Print the index of the matrix

move $a0, $t0

syscall

li $v0, 4 # Print 2Points

la $a0, 0($s0)

syscall

li $v0, 5 # Read the Integer

syscall

add $t3, $0, $v0 # Store the number given by the user into $t3

sw $v0, matrixElements($t9)

addi $t9, $t9, 4

sb $t3, 0($t2) # Store the number in the Matrix

addi $t2, $t2, 1 # Increase by 1 the Index, the Counter and the Temporal Counter ($s1)

addi $t0, $t0, 1

addi $s1, $s1, 1

bne $t4, $t0, read # If the counter it is not equal to the dimension, keep reading

ragain:

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

xor $t0, $t0, $t0 # Reinitialize Counter

bne $s1, $t5, read # Read Again; Now for Matrix B

intlz1:

la $t2, MATA # Reload the address of MATA (Matrix A) into $t2

la $t3, space # Load the address of Spaces into $t3

xor $s1, $s1, $s1 # Reinitialize Temporal Counter

la $t1, MATBT # Load the address of MATBT into $t1

trnpse:

lbu $t6, 9($t2) # Load B[i]

sb $t6, 0($t1) # Store B[i] into it's transpose (BT[i])

addi $t2, $t2, 3 # Offset for the Transpose

addi $t0, $t0, 3 # Increase counter in 3

addi $t1, $t1, 1 # Increase by 1 the index of BT[i]

bne $t0, $t4, trnpse # If the 3 numbers of a line are printed, go to the next one

addi $s1, $s1, 1 # Next Line

la $t2, MATA # Reload the address of MATA into $t2

xor $t0, $t0, $t0 # Reinitialize Counter

add $t2, $t2, $s1 # Increase the offset for $t2

bne $s1, 3, trnpse # If the 3 lines are not printed, repeat the process

#>>>>>>>>>>>>>>>>>>>>Printing Elements of Matrix A<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

printMatrixA:

addi $t0, $zero, 0

li $v0, 4 # Print Matrix A Message

la $a0, matrixA

syscall

matrixANewLine:

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

addi $t9, $zero, 0

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

printMatrixAElements:

beq $t0, 36, printMatrixB

lw $t8, matrixElements($t0)

addi $t0, $t0, 4

li $v0, 1

move $a0, $t8

syscall

li $v0, 4 # Print the spaces between numbers

la $a0, space

syscall

addi $t9, $t9, 1

beq $t9, 3, matrixANewLine # If all the numbers of the Matrix A haven't been printed, do all the same process

j printMatrixAElements

#>>>>>>>>>>>>>>>>>>>>Done Printing Elements of Matrix A<<<<<<<<<<<<<<<<<<<<<<<<<<<<

#>>>>>>>>>>>>>>>>>>>>Printing Elements of Matrix B<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

printMatrixB:

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

addi $t0, $zero, 36

li $v0, 4 # Print Matrix B Message

la $a0, matrixB

syscall

matrixBNewLine:

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

addi $t9, $zero, 0

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

printMatrixBElements:

beq $t0, 72, intlz2

lw $t8, matrixElements($t0)

addi $t0, $t0, 4

li $v0, 1

move $a0, $t8

syscall

li $v0, 4 # Print the spaces between numbers

la $a0, space

syscall

addi $t9, $t9, 1

beq $t9, 3, matrixBNewLine # If all the numbers of the Matrix B haven't been printed, do all the same process

j printMatrixBElements

#>>>>>>>>>>>>>>>>>>>>Done Printing Elements of Matrix B<<<<<<<<<<<<<<<<<<<<<<<<<<<<

intlz2:

li $v0, 4 # Call service in order to print an "End of Line" twice

la $a0, 0($s3)

syscall

la $t6, matrixC # Load the address of Message (Matrix C) into $t6

li $s2, -3 # MATA Offset Counter; Counter for Loop i

xor $s1, $s1, $s1 # Reinitialize Temporal Counter

li $v0, 4 # Print Message 2

la $a0, 0($t6)

syscall

loopi:

xor $s0, $s0, $s0 # Initialize counter for loop j

la $t1, MATBT # Reload the address of MATBT into $t1

addi $s2, $s2, 3 # Increase in 3 the counter for loop i

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

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

loopj:

beq $s0, $t4, loopi # If $s0 it's equal to $t4, go to loop i

la $t2, MATA # Reload the address of MATA into $t2

add $t2, $t2, $s2 # Increase the index of A[i] depending the current state of the process

xor $s4, $s4, $s4 # Reinitialize Sum for C[i]

xor $t0, $t0, $t0 # Reinitialize Counter for loop k

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

loopk:

lbu $t5, 0($t2) # Load A[i]

lbu $t6, 0($t1) # Load BT[i]

mul $t7, $t5, $t6 # $t7 = A[i] \* B[i]

add $s4, $s4, $t7 # sum = sum + $t7

addi $t2, $t2, 1 # Increase by 1 the indices of A[i] & BT[i];

addi $t1, $t1, 1

addi $t0, $t0, 1 # Increase by 1 the counter for loop k & the counter for loop j

addi $s0, $s0, 1

bne $t0, 3, loopk # If $t0 it's not equals to 3, go to loop k

addi $s1, $s1, 1 # Increase by 1 the counter of the index of MATC

la $t2, MATA # Reload the address of MATA into $t2

add $t2, $t2, $s1 # Increase the index of C[i]

sb $s4, 17($t2) # Store the value of $s4 into MATC

li $v0, 1 # Print number of C[i]

add $a0, $0, $s4

syscall

li $v0, 4 # Print the spaces between numbers

la $a0, 0($t3)

syscall

bne $s1, $t4, loopj # If all the numbers of the MATC haven't been printed, do all the same process

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

li $v0, 4 # Call service in order to print an "End of Line"

la $a0, 0($s3)

syscall

exit:

li $v0, 10 # Exit Program

syscall

1. Provide answers to the questions asked. Use the space below. [40]

A matrix is a rectangular arrangement of numbers into rows and columns. Each number in a matrix is referred to as a matrix element or entry. The dimensions of a matrix give the number of rows and columns of the matrix *in that order*. Since matrix A & B has 2 rows and 2 columns, they are called 2\*2 matrix for the first execution and since matrix A & B (In Second Program) has 3 rows and 3 columns, they are called a 3\*3 matrix for the second execution. In matrix multiplication, each entry in the product matrix is the dot product of a row in the first matrix and a column in the second matrix.

The three loops in iterative matrix multiplication can be arbitrarily swapped with each other without an effect on correctness or asymptotic running time. However, the order can have a considerable impact on practical performance due to the memory access patterns and cache use of the algorithm,which order is best also depending on whether the matrices are stored in row-major order, column-major order, or a mix of both.

1. The performance of a computer is determined by three key factors:
   * Instruction count
   * Clock cycle time
   * Clock cycles per instruction (CPI)
2. The compiler and the instruction set architecture determine the instruction count required for a given program
3. However, the implementation of the processor determines both the clock cycle time and the number of clock cycles per instruction
4. The core MIPS instructions includes:
   * Integer arithmetic-logical instructions
   * Memory-reference instructions
   * Branch instructions
5. Much of what needs to be done to implement these instructions is the same, independent of the exact class of instruction
6. For every instruction, the first two steps are identical:
   * Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory
   * Read one or two registers, using fields of the instruction to select the registers to read. For the *load word* instruction, we need to read only one register, but most other instructions require reading two registers
7. After these two steps, the actions required to complete the instruction depend on the instruction class
8. For each of the three instruction classes, the actions are largely the same, independent of the exact instruction - *memory-reference*, *arithmetic-logical*, and *branches*

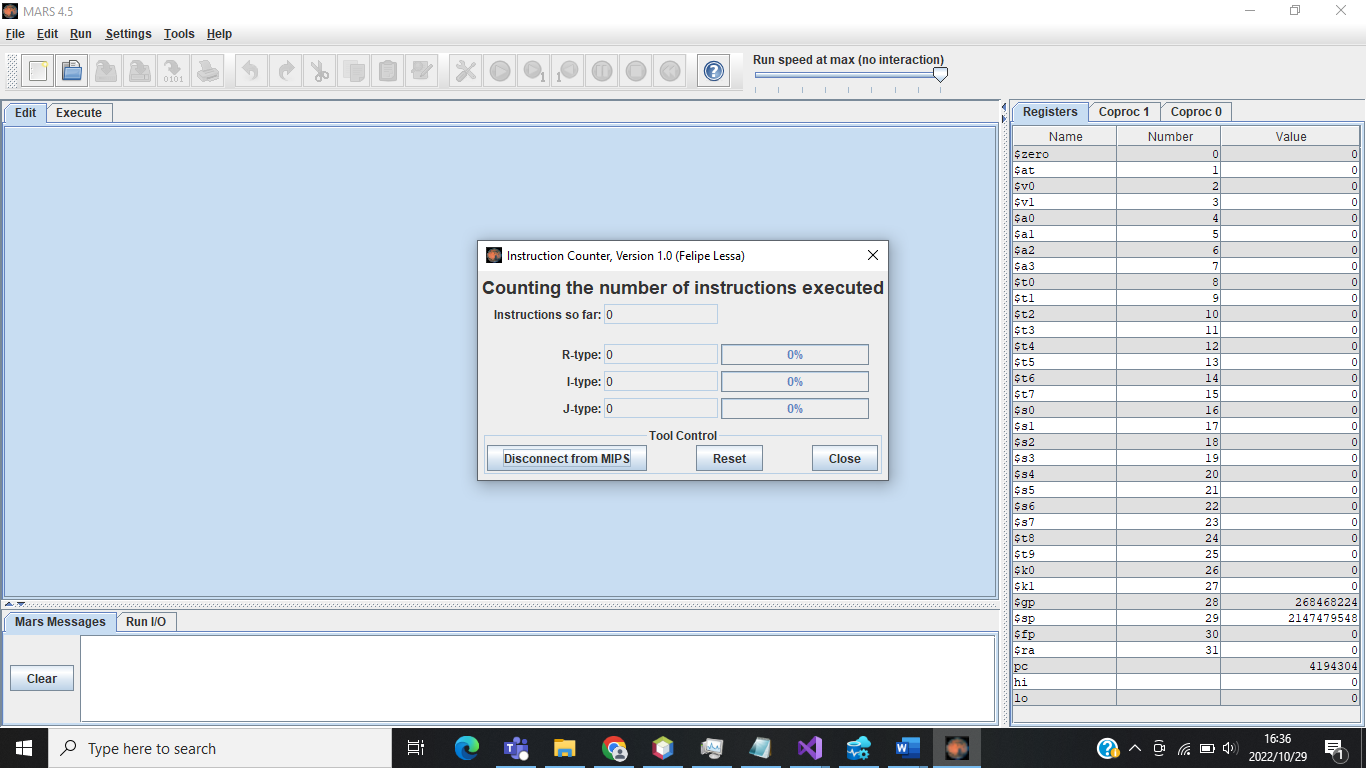
**Matrix\_Multiplication (2x2 vs 3x3)**

Before we could find if there is difference in the execution time for the two versions of Matrix Multiplication (2x2 & 3x3), we had to find out how many instructions each piece of code executed.

We went to MARS 4.5 Simulator, Opened the program to find the number of instructions executed as they will determine the execution time.

In MARS 4.5 Software, Go To: *Tools -> Instruction Counter -> Connect to MIPS*

This should open the following windows, and of course you must ensure that it is connected.



After it has been connected, Assemble, and execute the program you want to find the number of instructions executed.

For **Matrix\_Multipliction\_2x2**, the instructions executed were: 598

R-type: 181

I-type: 413

J-type: 4

**TOTAL: 598**

For **Matrix\_Multipliction\_3x3**, the instructions executed were: 1297

R-type: 378

I-type: 907

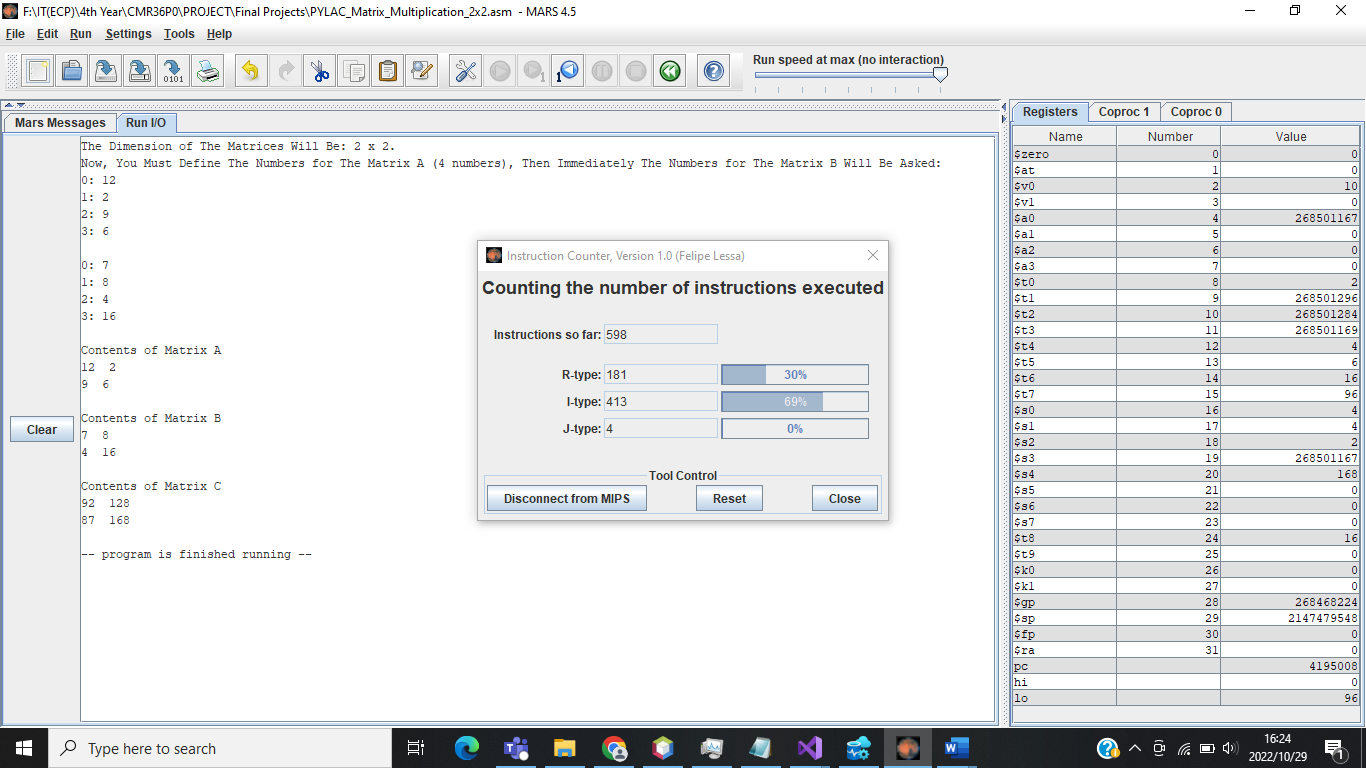
J-type: 12

**TOTAL: 1297**

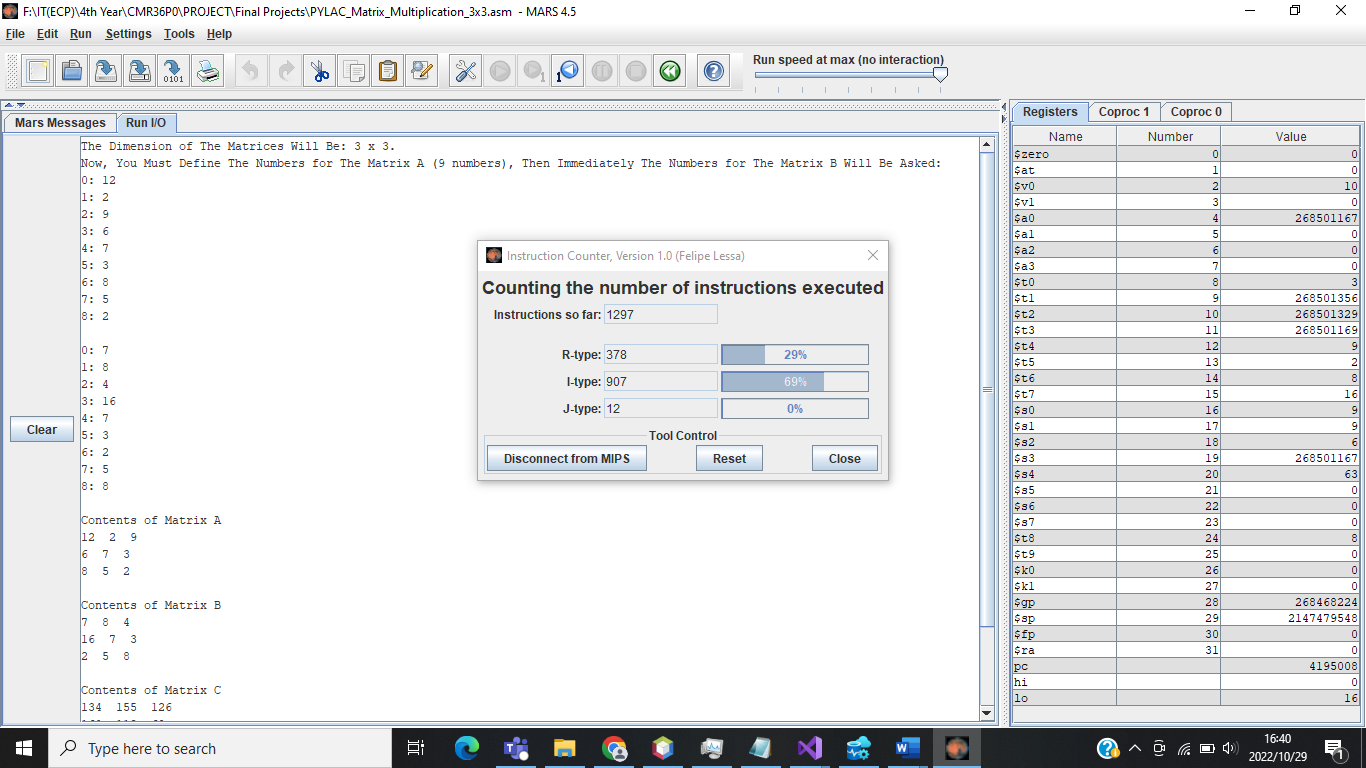
**NB: Before Finding the instructions executed for each program, you must reset the instructions counter tool, otherwise it will keep accumulating them resulting in wrong instructions counting.**

**(Use once, then Reset for the next use)**

Below are the screenshots that supports the above statements

**Matrix\_Multiplication\_2x2**

**Matrix\_Multiplication\_3x3**

The performance of a piece of code is based on execution time. Execution time is the total time a CPU spends computing a task. Calculation of execution time is based on three main aspects:

1. The number of instructions in the program

2. The cycles per instruction (CPI) and

3. The cycle time. The formula is:

*(Execution time = Number of instructions in the program \* CPI \* Clock time)*

**Calculation of CPI (Cycles Per Instruction)**

Load 5 cycles

Store 4 cycles

R-type 4 cycles

Branch 3 cycles

Jump 3 cycles

The programs have:

50% R-type instructions

10% load instructions

20% store instructions

8% branch instructions

2% jump instructions

**THEN:** CPI = (4x50 + 5x10 + 4x20 + 3x8 + 3x2) / 100

CPI = 3.6

Code 1 (**Matrix\_Multiplication\_2x2)**:

The first piece of code accepts content of matrix A and matrix B and stores product in matrix C. All three matrices have 2 rows and 2 columns.

Ex1:

Number of Instructions: 598

Execution Time = 598 \* 3,6 \*

Execution Time = 119 ms

Code 2 (**Matrix\_Multiplication\_3x3)**:

The second piece of code accepts content of matrix A and matrix B and stores product in matrix C. However, all three matrices have 3 rows and 3 columns now. The addition of matrices means more instructions will have to be implemented to carry out execution. Ex2 below illustrates a higher number of instructions than in Ex1 and the effects on the execution time can be seen.

Ex2:

Number of Instructions: 1297

Execution Time = 598 \* 3,6 \*

Execution Time = 259 ms

In Conclusion:

Both codes execute on the same machine and on the same compiler. The only difference is in the Number of Instructions. Code 2 contains more instructions than Code 1. Therefore, execution time in code 1 is less (faster) than the execution time in code 2. In terms of performance, Code 1 uses less memory whereas Code 2 uses more memory