UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
INE-DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
INE5411 - ORGANIZAÇÃO DE COMPUTADORES I

PEDRO TAGLIALENHA (22203674)
ERIK ORSOLIN DE PAULA (22102195)

RELATÓRIO LABORATÓRIO 2

FLORIANÓPOLIS
2023

1- Introdução

No laboratório 2 da disciplina de Organização de Computadores, realizamos uma atividade prática utilizando a linguagem Assembly para o processador MIPS no simulador MARS. O objetivo foi aplicar os conceitos teóricos aprendidos sobre operações e manipulações de matrizes e aprofundar conhecimentos sobre a gestão de memória e representação ASCII.

Na primeira tarefa, desenvolvemos um programa para calcular o produto de uma matriz A com a transposta da matriz B, armazenando o resultado na memória de dados.

O segundo desafio expandiu o primeiro ao requerer o armazenamento da matriz resultante em um arquivo .txt, com os elementos convertidos para representações ASCII, permitindo a visualização humana no arquivo criado.

Finalmente, na terceira tarefa, evoluímos o código inicial para incluir procedimentos específicos para a multiplicação de matrizes e a criação de uma matriz transposta, praticando a modularização do código em Assembly.

Esta atividade permitiu não apenas a aplicação prática dos conceitos teóricos estudados, mas também um aprofundamento no uso da linguagem Assembly, proporcionando uma base sólida para trabalhos futuros nesta disciplina.

2- Inicialização e carga de dados

```
.data
    .align 2
    A: .word 1 2 3 0 1 4 0 0 1
    .align 2
    B: .word 1 -2 5 0 1 -4 0 0 1
    .align 2
    C: .word 0 0 0 0 0 0 0 0 0
```

Três matrizes 3x3 são definidas no segmento de dados, alinhadas a 2 bytes. As matrizes A e B recebem valores inicializados, enquanto C é preenchida com zeros.

3- Funções de Assembly Usadas

- **.align** – Usado para alinhar os dados na memória, facilitando o acesso a eles.
- **.word** – Define palavras de 32 bits na memória.
- **.asciiz** - Esta diretiva é usada para declarar uma string, terminada por um null
- .**syscall** - Usada para fazer chamadas de sistema para realizar operações de E/S e outras funções do sistema operacional.
- **la** – Carrega o endereço de uma variável no registrador.
- **li** - É uma instrução que carrega um valor imediato (constante) em um registrador.
- **lw** – Carrega uma palavra da memória no registrador.
- **sw** – Armazena uma palavra do registrador na memória.
- **move** – Move o conteúdo de um registrador para outro.
- **mul** – Multiplica o conteúdo de dois registradores e armazena o resultado em um registrador.
- **add** e **addi** – Soma valores de registradores ou um valor imediato a um valor de registrador.

- **beq** – Desvio condicional se os valores dos dois registradores forem iguais.
- **j** – Instrução de salto incondicional para um rótulo.

4- Códigos desenvolvidos

Para o laboratório 2 foram desenvolvidos quatro códigos em assembly, duas versões para a questão 1 e uma para as demais. O código 1 foi a implementação inicial que apresentou-se com diversos problemas e por isso foi realizado um segundo código para solucionar corretamente a questão 1.

4.1- Questão 1

Código 1: Protótipo inicial

```
.data
    .align 3
    A: .word 1 2 3 0 1 4 0 0 1
    .align 3
    B: .word 1 -2 5 0 1 -4 0 0 1
    .align 3
    C: .word 0 0 0 0 0 0 0 0 0

.text

    la $s0, A  # carregado o endereço base da matriz A
    la $s1, B  # carregado o endereço base da matriz B
    la $s3, C

    # troca de 1,2 para 2,1
    lw $t2, 4($s1) #  carregado no registrador t2 o valor 1,2 da matriz B
    lw $t3, 12($s1)
    sw $t3, 4($s1)
    sw $t2, 12($s1)

    # troca de 1,3 para 3,1
    lw $t2, 8($s1)
    lw $t3, 24($s1)
    sw $t3, 8($s1)
    sw $t2, 24($s1)

    # troca de 2,3 para 3,2
    lw $t2, 20($s1)
    lw $t3, 28($s1)
    sw $t3, 20($s1)
    sw $t2, 28($s1)

    ################################################
    # Multiplicações #
    # #
    ################################################

    #mult para o 1,1
    lw $t0, 0($s0)
    lw $t1, 0($s1)
    mult $t0, $t1
    mflo $t2
    sw $t2, 0($s3)

    #mult para o 1,2
    lw $t0, 4($s0)
    lw $t1, 4($s1)
    mult $t0, $t1
    mflo $t2
    sw $t2, 4($s3)

    #mult para o 1,3
    lw $t0, 8($s0)
    lw $t1, 8($s1)
    mult $t0, $t1
```

```
    mflo $t2
    sw $t2, 8($s3)

    #mult para o 2,1
    lw $t0, 12($s0)
    lw $t1, 12($s1)
    mult $t0, $t1
    mflo $t2
    sw $t2, 12($s3)

    #mult para o 2,2
    lw $t0, 16($s0)
    lw $t1, 16($s1)
    mult $t0, $t1
    mflo $t2
    sw $t2, 16($s3)

    #mult para o 2,3
    lw $t0, 20($s0)
    lw $t1, 20($s1)
    mult $t0, $t1
    mflo $t2
    sw $t2, 20($s3)

    #mult para o 3,1
    lw $t0, 24($s0)
    lw $t1, 24($s1)
    mult $t0, $t1
    mflo $t2
    sw $t2, 24($s3)

    #mult para o 3,2
    lw $t0, 28($s0)
    lw $t1, 28($s1)
    mult $t0, $t1
    mflo $t2
    sw $t2, 28($s3)

    #mult para o 3,3
    lw $t0, 32($s0)
    lw $t1, 32($s1)
    mult $t0, $t1
    mflo $t2
    sw $t2, 32($s3)

     li $v0, 1
     lw $a0, 0($s3)
     syscall
```

Fonte: Elaborado por autores(2023)

Na primeira implementação da solução da questão, cada etapa do processo de multiplicação de matrizes foi realizada manualmente, incluindo a transposição e a multiplicação individual dos elementos das matrizes A e B utilizando em uma sequência bastante extensa de operações de load-word (lw), multiplication (mult) e store-word (sw), efetuou-se manualmente cada multiplicação dos elementos correspondentes e armazenamento do resultado na matriz C.

Este processo de correção evidenciou uma limitação significativa da primeira abordagem adotada, que era a extensão desnecessariamente longa do código, derivada da realização manual de cada operação individualmente, sem recorrer à implementação de loops ou funções que poderiam simplificar e encurtar significativamente o código, facilitando tanto a escrita quanto a manutenção do mesmo.Também foi observado que na versão inicial do código continha um erro fundamental na execução da multiplicação das matrizes, que foi corrigido na segunda versão do código.

Código 2: Segunda implementação da Questão 1

```
.data
    .align 2
    A: .word 1 2 3 0 1 4 0 0 1
    .align 2
    B: .word 1 -2 5 0 1 -4 0 0 1
    .align 2
    C: .word 0 0 0 0 0 0 0 0 0

.text

    la $s0, A  # carregado o endereço base da matriz A
    la $s1, B  # carregado o endereço base da matriz B
    la $s2, C  # carregando o endereço base da matriz resultante


    # $t2 = linha
    # $t3 = coluna
    # $t4 = k
    move $t2, $zero # inicializando com zero o registrador que será usado para controle do laço da linha
    move $t3, $zero # inicializando com zero o registrador que será usado para controle do laço da coluna
    move $t4, $zero # inicializando com zero o registrador que será usado para controle do laço das operações


    ##############################################
         # Calculando a transposta de B #
    ##############################################

    # troca de 1,2 para 2,1
    lw $t0, 4($s1) #  carregado no registrador t2 o valor 1,2 da matriz B
    lw $t1, 12($s1)
    sw $t1, 4($s1)
    sw $t0, 12($s1)

    # troca de 1,3 para 3,1
    lw $t0, 8($s1)
    lw $t1, 24($s1)
    sw $t1, 8($s1)
    sw $t0, 24($s1)

    # troca de 2,3 para 3,2
    lw $t0, 20($s1)
    lw $t1, 28($s1)
    sw $t1, 20($s1)
    sw $t0, 28($s1)



    ##############################################
                  # Multiplicações #
    ##############################################


    loop_linha:
    beq $t2, 3, fim_loop_linha # permacene no loop linha enquanto $t2 for menor do que 3
           move $t3, $zero   # reinicia o contador de colunas

           loop_coluna:
           beq $t3, 3, fim_loop_coluna  # permacene no loop coluna enquanto $t3 for menor do que 3
                    move $t5, $zero   # $t5 será usado para a soma das multiplicações
                    move $t4, $zero   # reinicia o contador de k para zero

                    loop_k:
                    beq $t4, 3, fim_loop_k    # permacene no loop k enquanto $t4 for menor do que 3
                    move $s4, $zero            # $s4 será usado para armazenar a multiplicação da vez

                             # Calculo do endereço A[linha][k]
                             mul $t7, $t2, 12   # $t7 = linha * 12 (calculando o deslocando das linhas)
                             mul $t8, $t4, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                             add $t7, $t7, $t8  # $t7 = $t7 + k * 4
                             add $t7, $t7, $s0  # $t7 = $t7 + endereço base da matriz A
```

```
                        lw $t6, 0($t7)    # $t6 = A[linha][k]


                        # Calculo do endereço B[k][coluna]
                        mul $t7, $t4, 12   # $t7 = k * 12 (percorrendo a coluna)
                        mul $t8, $t3, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                        add $t7, $t7, $t8  # $t7 = $t7 + coluna * 4
                        add $t7, $t7, $s1  # $t7 = $t7 + endereço base da matriz B
                        lw $t9, 0($t7)    # $t9 = B[k][coluna]


                        mul $s4, $t6, $t9  # A[linha][k] * B[k][coluna]

                        add $t5, $t5, $s4  # $t5 contém o somatório das multiplicações

                addi $t4, $t4, 1 # $t4 += 1
                j loop_k
                fim_loop_k:

                # Calculo do endereço C[linha][coluna]
                mul $t7, $t2, 12   # $t7 = $t2 * 12 (calculando o deslocamento de linhas)
                mul $t8, $t3, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                add $t7, $t7, $t8  # # $t7 = $t7 + coluna * 4
                add $t7, $t7, $s2  # $t7 = $t7 + endereço base da matriz C
                sw  $t5, 0($t7)   # escrevendo o resultado C[i][j] na matriz C


        addi $t3, $t3, 1 # $t3 += 1
        j loop_coluna
        fim_loop_coluna:

addi $t2, $t2, 1 # $t2 += 1
j loop_linha
fim_loop_linha:
```

Fonte: Elaborado por autores(2023)

Na segunda versão da implementação, implementamos uma estruturação mais direta e eficiente do código, que é dividido em duas grandes etapas: a transposição da matriz B e a multiplicação das matrizes A e B. Na primeira etapa, a matriz B é transposta por meio da troca direta dos valores entre as posições simétricas em relação à diagonal principal.

A grande melhoria se dá na etapa de multiplicação das matrizes, onde foi implementada uma estrutura de três loops aninhados para percorrer as linhas, colunas e realizar a somatória necessária na obtenção de cada elemento da matriz resultado. Esse novo arranjo promove uma maior automação do processo, evitando a repetição desnecessária de linhas de código

Os laços de repetição ("loops") são utilizados para iterar através das linhas e colunas das matrizes A e B, e para calcular os elementos da matriz resultado C. Os desvios condicionais, como **beq** são utilizados para controlar o fluxo dos loops, verificando se o contador atual atingiu o limite especificado (neste caso, 3) e, consequentemente, se o loop deve terminar. Dentro do loop mais interno, as operações de multiplicação e adição são realizadas para calcular o valor de um elemento específico da matriz C. Após cada iteração, os contadores são incrementados utilizando **addi** até que se atinja a condição de término, momento no qual o loop é encerrado e o programa avança para o próximo conjunto de iterações.

Agora, a multiplicação é feita de maneira correta, respeitando a regra da multiplicação de matrizes, onde o elemento C[i][j] é obtido através da somatória da multiplicação dos elementos correspondentes das linhas de A pelas colunas de B.

4.2- Questão 2

Código 3: Solução Questão 2

```
.data
    .align 2
    A: .word 1 2 3 0 1 4 0 0 1

    .align 2
    B: .word 1 -2 5 0 1 -4 0 0 1

    .align 2
    C:  .word 0 0 0 0 0 0 0 0 0

    String: .asciiz "0"

    Negativo: .asciiz "  -"

    Espaco: .asciiz " "

    quebra_linha: .asciiz "  \n"

    CaminhoArquivo: .asciiz "/home/erik/Documentos/matriz.txt"

.text

    la $s0, A  # carregado o endereço base da matriz A
    la $s1, B  # carregado o endereço base da matriz B
    la $s2, C  # carregando o endereço base da matriz resultante


    # $t2 = linha
    # $t3 = coluna
    # $t4 = k
    move $t2, $zero # inicializando com zero o registrador que será usado para controle do laço da linha
    move $t3, $zero # inicializando com zero o registrador que será usado para controle do laço da coluna
    move $t4, $zero # inicializando com zero o registrador que será usado para controle do laço das operações


    ##############################################
         # Calculando a transposta de B #
    ##############################################

    # troca de 1,2 para 2,1
    lw $t0, 4($s1) #  carregado no registrador t2 o valor 1,2 da matriz B
    lw $t1, 12($s1)
    sw $t1, 4($s1)
    sw $t0, 12($s1)

    # troca de 1,3 para 3,1
    lw $t0, 8($s1)
    lw $t1, 24($s1)
    sw $t1, 8($s1)
    sw $t0, 24($s1)

    # troca de 2,3 para 3,2
    lw $t0, 20($s1)
    lw $t1, 28($s1)
    sw $t1, 20($s1)
    sw $t0, 28($s1)



    ##############################################
                  # Multiplicações #
    ##############################################

    li $v0, 13     # código syscall para abrir arquivo
    la $a0, CaminhoArquivo
    li $a1, 1      # abrir para escrita
    syscall
```

```
    move $s6, $v0   # descritor salvo em $s6

    loop_linha:
    beq $t2, 3, fim_loop_linha # permacene no loop linha enquanto $t2 for menor do que 3
          move $t3, $zero   # reinicia o contador de colunas

          loop_coluna:
          beq $t3, 3, fim_loop_coluna  # permacene no loop coluna enquanto $t3 for menor do que 3
                    move $t5, $zero   # $t5 será usado para a soma das multiplicações
                    move $t4, $zero   # reinicia o contador de k para zero

                    loop_k:
                    beq $t4, 3, fim_loop_k    # permacene no loop k enquanto $t4 for menor do que 3
                    move $s4, $zero            # $s4 será usado para armazenar a multiplicação da vez

                              # Calculo do endereço A[linha][k]
                              mul $t7, $t2, 12   # $t7 = linha * 12 (calculando o deslocando das linhas)
                              mul $t8, $t4, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                              add $t7, $t7, $t8  # $t7 = $t7 + k * 4
                              add $t7, $t7, $s0  # $t7 = $t7 + endereço base da matriz A
                              lw $t6, 0($t7)    # $t6 = A[linha][k]


                              # Calculo do endereço B[k][coluna]
                              mul $t7, $t4, 12   # $t7 = k * 12 (percorrendo a coluna)
                              mul $t8, $t3, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                              add $t7, $t7, $t8  # $t7 = $t7 + coluna * 4
                              add $t7, $t7, $s1  # $t7 = $t7 + endereço base da matriz B
                              lw $t9, 0($t7)    # $t9 = B[k][coluna]


                              mul $s4, $t6, $t9  # A[linha][k] * B[k][coluna]

                              add $t5, $t5, $s4  # $t5 contém o somatório das multiplicações

                    addi $t4, $t4, 1 # $t4 += 1
                    j loop_k
                    fim_loop_k:

                    # Calculo do endereço C[linha][coluna]
                    mul $t7, $t2, 12   # $t7 = $t2 * 12 (calculando o deslocamento de linhas)
                    mul $t8, $t3, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                    add $t7, $t7, $t8  # # $t7 = $t7 + coluna * 4
                    add $t7, $t7, $s2  # $t7 = $t7 + endereço base da matriz C

                    bgez $t5, maior_igual_zero # se o número a ser escrito for positivo vai pro label

                    la $t9, Negativo  # lendo endereço do caracter menos
                    addi $t9, $t9, 2  # somando 2 (em byte) ao endereço do caracter
                    # Isso está sendo feito pois no endereço original estava sendo gravado dois caracteres NULL
(/00) antes do caracter menos

                    li $v0, 15        # código syscall para escrita em arquivo
                    move $a0, $s6  # descritor em $a0
                    move $a1, $t9  # endereço do caracter de sinal
                    li $a2, 1         # quantidade de caracteres
                    syscall
                    abs $t5, $t5

                    j calculo

                    maior_igual_zero:
                    li $v0, 15        # código syscall para escrita em arquivo
                    move $a0, $s6   # descritor em $a0
                    la $a1, Espaco  # endereço do caracter de sinal
                    li $a2, 1         # quantidade de caracteres
                    syscall

                    calculo:
                    div $t0, $t5, 10  # parte da dezena vai para $t0
                    mfhi $t1      # parte da unidade vai para $t1
```

```
                addi $t0, $t0, 48  # convertendo a dezena para sua representação em ASCII
                sw $t0, String    # gravando o valor na String

                li $v0, 15        # código syscall para escrita em arquivo
                move $a0, $s6   # descritor em $a0
                la $a1, String  # endereço da String
                li $a2, 1         # quantidade de caracteres
                syscall

                addi $t1, $t1, 48 # convertendo a unidade para sua representação ASCII
                sw $t1, String  # gravando o valor unidade na String

                li $v0, 15        # código syscall para escrita em arquivo
                move $a0, $s6   # descritor em $a0
                la $a1, String  # endereço da String
                li $a2, 1         # quantidade de caracteres
                syscall

                li   $v0, 15      # Comando para escrita
                move $a0, $s6     # descritor em $a0
                la   $a1, Espaco   # Carrega " "
                li   $a2, 1       # Um caracter
                syscall

                sw  $t5, 0($t7)   # escrevendo o resultado C[i][j] na matriz C

        addi $t3, $t3, 1 # $t3 += 1
        j loop_coluna
        fim_loop_coluna:

        li $v0, 15                # código syscall para escrita em arquivo
        move $a0, $s6             # descritor em $a0
        la $a1, quebra_linha  # endereço de quebra_linha
        li $a2, 3                 # quantidade de caracteres
        syscall

addi $t2, $t2, 1 # $t2 += 1
j loop_linha
fim_loop_linha:

li   $v0, 16            # fechando arquivo
move $a0, $s6
syscall
```

Fonte: Elaborado por autores(2023)

Ao elaborarmos a solução para armazenar os elementos da matriz resultante em um arquivo .txt com representações ASCII, nós nos deparamos com o desafio de representar os números, tanto positivos quanto negativos, em um formato legível para humanos.

Para abordar este problema, desenvolvemos um sistema que primeiro verifica se o número é negativo. Se for, nós inserimos um sinal de menos ('-') no arquivo antes do número, garantindo que ele seja claramente identificado como negativo. Para fazer isso, utilizamos o comando bgez para desviar para um rótulo específico (maior_igual_zero) se o número for maior ou igual a zero. Caso contrário, o sinal de menos é inserido no arquivo.

Após garantir que o sinal esteja correto, procedemos com a conversão dos números para representações ASCII. Dividimos o número por 10, de modo que o quociente representasse a dezena e o resto representasse a unidade. Esses valores foram então convertidos para suas respectivas representações ASCII, somando-se 48 (o valor ASCII de '0') a cada um deles. Este processo foi efetuado utilizando as instruções div, mfhi, e uma adição imediata (addi) de 48. Posteriormente, armazenamos esses valores convertidos em ASCII em uma string, a qual é escrita no arquivo.

Optamos por escrever cada elemento da matriz individualmente, o que nos permitiu inserir espaços e quebras de linha para formatar a matriz de forma legível no arquivo de texto, garantindo assim que qualquer pessoa possa abrir o arquivo e visualizar claramente a matriz resultante.

Durante o desenvolvimento do nosso código, enfrentamos uma série de dificuldades, a principal estava relacionada ao comportamento anômalo do código dependendo do sistema operacional em que o mars estava sendo executado. Nos deparamos com um problema persistente onde, em máquinas que rodavam o sistema operacional Windows, os símbolos de negativo não estavam sendo devidamente escritos no arquivo de texto, enquanto no Linux, o que aparecia era o código ASCII de null "\00" em vez do desejado sinal de negativo.

Após muitas tentativas de solucionar o problema, percebemos que o problema residia na forma como estávamos declarando o símbolo de negativo no nosso código. Chegamos a uma resolução ao redefinir a declaração do símbolo de negativo para incluir dois espaços em branco antes dele. Depois manipulamos a memória para pegarmos o símbolo de negativo desejado como nos seguintes trecho de código do Código 3.

```
Negativo: .asciiz "  -"


la $t9, Negativo  # lendo endereço do carácter -
addi $t9, $t9, 2  # somando 2 (em byte) ao endereço do carácter
```

4.3- Questão 3

Código 4: Solução Questão 4

```
.data
    .align 2
    A: .word 1 2 3 0 1 4 0 0 1
    .align 2
    B: .word 1 -2 5 0 1 -4 0 0 1
    .align 2
    C: .word 0 0 0 0 0 0 0 0 0

.text
    .main:
          jal PROC_TRANS
          jal PROC_MUL
          li $v0, 10 # informa que o programa main acabou
          syscall




    PROC_TRANS:
          la $s0, A  # carregado o endereço base da matriz A
          la $s1, B  # carregado o endereço base da matriz B
          la $s2, C  # carregando o endereço base da matriz resultante


          # $t2 = linha
          # $t3 = coluna
          # $t4 = k
          move $t2, $zero # inicializando com zero o registrador que será usado para controle do laço da linha
          move $t3, $zero # inicializando com zero o registrador que será usado para controle do laço da coluna
          move $t4, $zero # inicializando com zero o registrador que será usado para controle do laço das
operações


          # troca de 1,2 para 2,1
          lw $t0, 4($s1) #  carregado no registrador t2 o valor 1,2 da matriz B
          lw $t1, 12($s1)
```

```
        sw $t1, 4($s1)
        sw $t0, 12($s1)

        # troca de 1,3 para 3,1
        lw $t0, 8($s1)
        lw $t1, 24($s1)
        sw $t1, 8($s1)
        sw $t0, 24($s1)

        # troca de 2,3 para 3,2
        lw $t0, 20($s1)
        lw $t1, 28($s1)
        sw $t1, 20($s1)
        sw $t0, 28($s1)

        jr $ra   # volta para onde a função foi chamada



PROC_MUL:
        loop_linha:
        beq $t2, 3, fim_loop_linha # permacene no loop linha enquanto $t2 for menor do que 3
                move $t3, $zero   # reinicia o contador de colunas

                loop_coluna:
                beq $t3, 3, fim_loop_coluna  # permacene no loop coluna enquanto $t3 for menor do que 3
                        move $t5, $zero   # $t5 será usado para a soma das multiplicações
                        move $t4, $zero   # reinicia o contador de k para zero

                        loop_k:
                        beq $t4, 3, fim_loop_k    # permacene no loop k enquanto $t4 for menor do que 3
                        move $s4, $zero            # $s4 será usado para armazenar a multiplicação da vez

                                # Calculo do endereço A[linha][k]
                                mul $t7, $t2, 12   # $t7 = linha * 12 (calculando o deslocando das linhas)
                                mul $t8, $t4, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                                add $t7, $t7, $t8  # $t7 = $t7 + k * 4
                                add $t7, $t7, $s0  # $t7 = $t7 + endereço base da matriz A
                                lw $t6, 0($t7)    # $t6 = A[linha][k]


                                # Calculo do endereço B[k][coluna]
                                mul $t7, $t4, 12   # $t7 = k * 12 (percorrendo a coluna)
                                mul $t8, $t3, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                                add $t7, $t7, $t8  # $t7 = $t7 + coluna * 4
                                add $t7, $t7, $s1  # $t7 = $t7 + endereço base da matriz B
                                lw $t9, 0($t7)    # $t9 = B[k][coluna]


                                mul $s4, $t6, $t9  # A[linha][k] * B[k][coluna]

                                add $t5, $t5, $s4  # $t5 contém o somatório das multiplicações

                        addi $t4, $t4, 1 # $t4 += 1
                        j loop_k
                        fim_loop_k:

                        # Calculo do endereço C[linha][coluna]
                        mul $t7, $t2, 12   # $t7 = $t2 * 12 (calculando o deslocamento de linhas)
                        mul $t8, $t3, 4   # $t8 = coluna * 4 (calculando o deslocamento de colunas)
                        add $t7, $t7, $t8  # # $t7 = $t7 + coluna * 4
                        add $t7, $t7, $s2  # $t7 = $t7 + endereço base da matriz C
                        sw  $t5, 0($t7)   # escrevendo o resultado C[i][j] na matriz C


                addi $t3, $t3, 1 # $t3 += 1
                j loop_coluna
                fim_loop_coluna:

        addi $t2, $t2, 1 # $t2 += 1
        j loop_linha
```

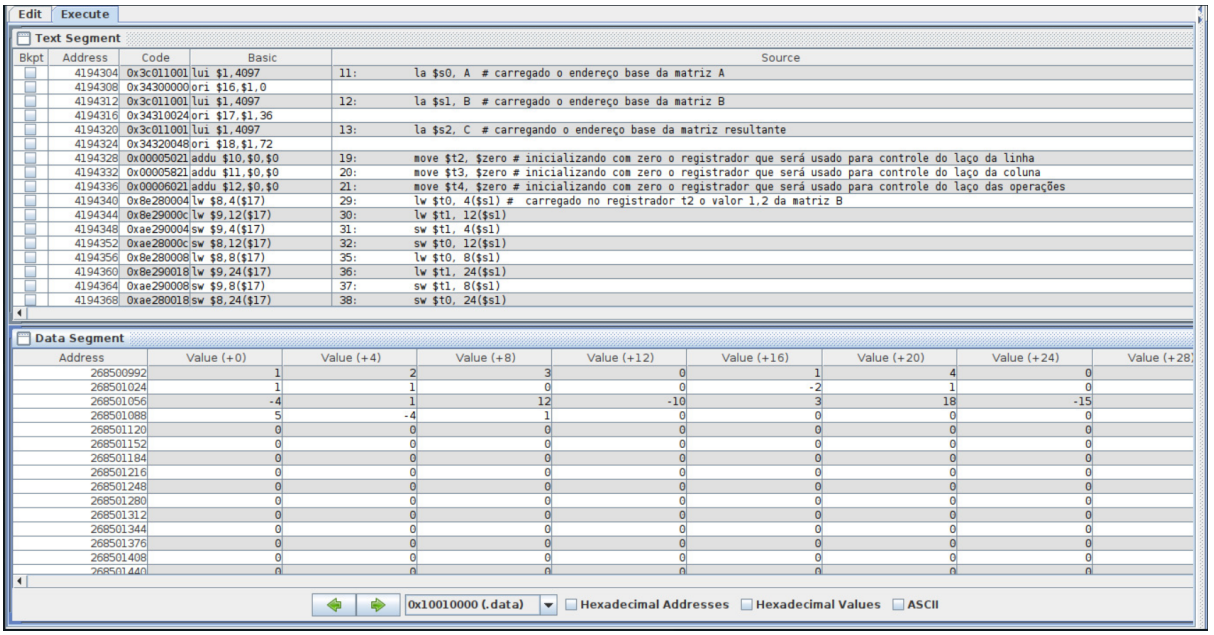
```
        fim_loop_linha:

        jr $ra # volta pra onde a função foi chamada
```

Fonte: Elaborado por autores(2023)

Enquanto o código 1 coloca tudo em um bloco de código, o código 3 divide as operações em procedimentos separados, o que facilita a leitura, manutenção e potencial reutilização dessas partes do código em outros programas. Portanto, o código 3 segue melhor as boas práticas de programação comparado ao código 1.

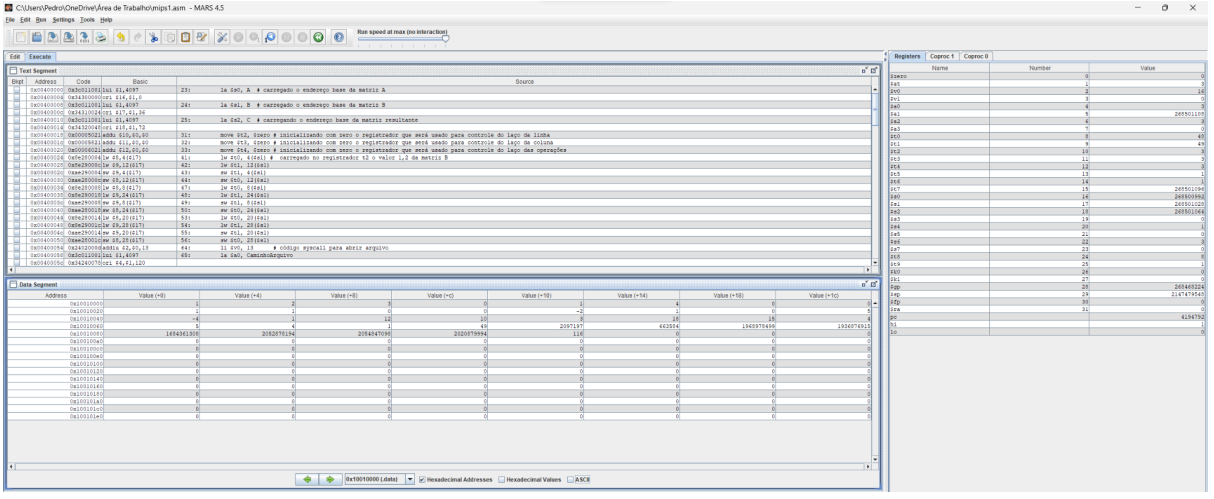
5- saídas dos códigos:

Figura 1: Saída do código 1



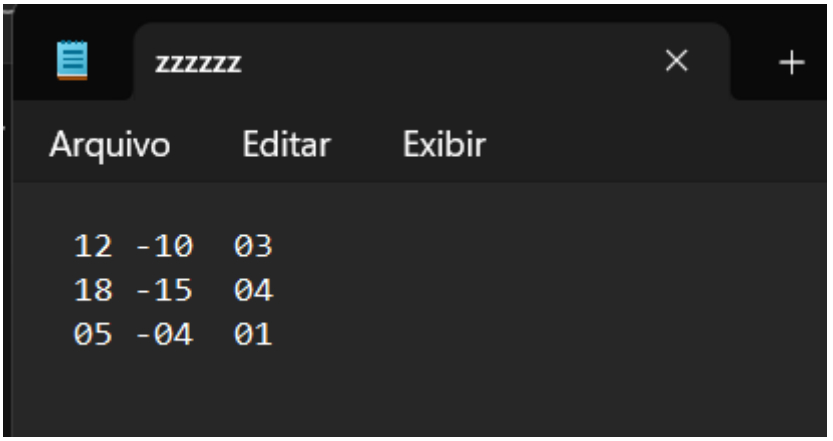
Fonte: Elaborado por autores(2023)

Figura 2: Saída do código 2



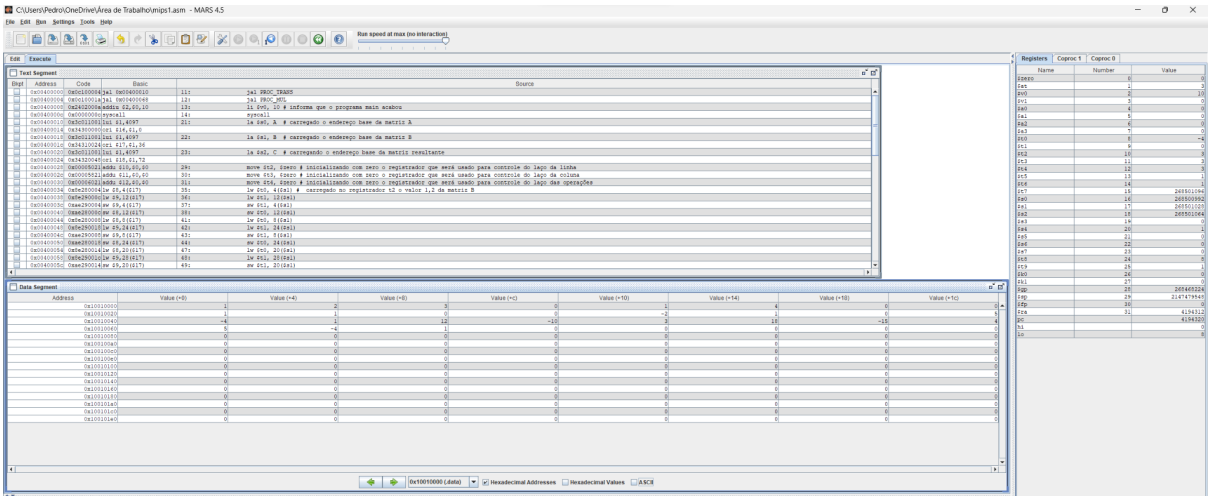
Fonte: Elaborado por autores(2023)

Figura 3: Arquivo de texto gerado pelo código 2



Fonte: Elaborado por autores(2023)

Figura 4: Saída do código 3



Fonte: Elaborado por autores(2023)

6- Conclusão

Ao longo deste laboratório, alcançamos sucesso no desenvolvimento e resolução das questões propostas, 1, 2 e 3. Através da execução das tarefas, fomos capazes de explorar e aprofundar nosso entendimento nos conteúdos ministrados.