UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS TRINDADE
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO CIÊNCIAS DA COMPUTAÇÃO

Rita Louro Barbosa – 22203157
Bianca Mazzuco Verzola - 22202621

INE5411 – Organização de Computadores I

Laboratório 02

● 1. Introdução

Os códigos desenvolvidos tomaram como base as informações dos módulos 1 a 9 de conteúdo disponibilizados no Moodle pelo professor. Também foram utilizadas como base as informações vistas em aulas práticas e teóricas.

● 2. Questão 1

2.1 Apresentação da questão

A questão 1 solicita um programa em Assembly do MIPS, a ser executado no simulador MARS que encontre a matriz resultante do produto A.Bt. e armazene a matriz resultante na memória de dados.

2.2 Estrutura do código

Para a implementação em Assembly das operações requisitadas, foram utilizadas instruções vistas em sala de aula, como as aritméticas, de transferência de dados e de desvio condicional.

Inicialmente, foram inicializadas na memória as matrizes A e B que posteriormente foram utilizadas como fonte para as operações. As matrizes B transposta e C também foram inicializadas na memória com todos os elementos iguais a 0 para que possam ser modificados depois.

Em seguida, foram carregados os endereços do primeiro elemento (endereço inicial da matriz) das matrizes inicializadas na memória em registradores de salvamento.

Para melhor visualização da explicação a seguir, estas são as posições (que serão chamadas de índice) dos elementos da matriz em relação ao endereço inicial da matriz:

0 4 8

12 16 20

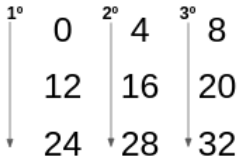
24 28 32

Para a transposição da matriz B, foram utilizados 2 loops (um externo e um interno):

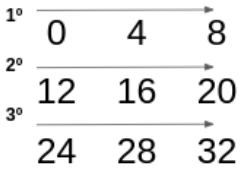
- o loop externo, chamado de Loop_coluna, percorre todas as colunas de B e, para controlar o seu número de iterações, é utilizado o registrador $s4 que terá o valor da posição do primeiro elemento da coluna em relação ao endereço de B, ou seja, valerá 0, 4 ou 8
- o loop interno, chamado de Loop_linha, percorre todas as linhas de B e, para controlar o seu número de iterações, é utilizado o registrador $s5 que valerá 0, 12 ou 24, dependendo de que linha de B que está sendo percorrida (0 para a primeira linha, 12 para a segunda e 24 para a terceira).

Dentro do Loop_linha, será feito o carregamento de um elemento de B, o qual poderá ser acessado por meio da soma de $s4 e $s5 com o endereço inicial de B. Esse elemento será colocado na matriz B transposta, a qual foi salva na memória para que pudesse ocorrer esses loops os quais vão incrementando em 4 posições o endereço do elemento de Bt a ser salvo.

Resumindo, matriz B é percorrida desta forma:



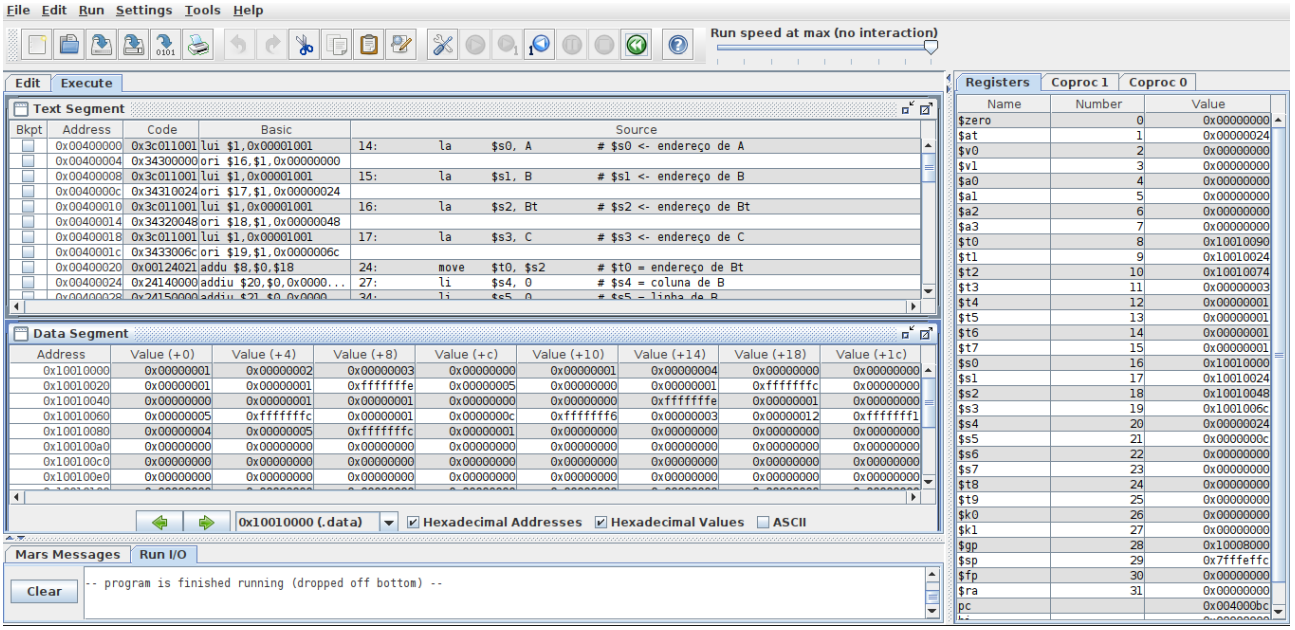
Enquanto a B transposta é salva desta forma:



Após isso, foi feita a multiplicação de A e Bt, na qual 3 loops foram utilizados:

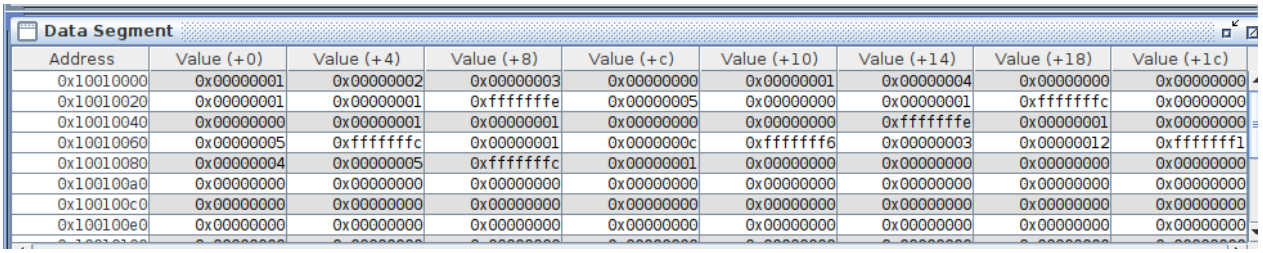
- o loop externo, chamado de Loop_linhas_A, percorre todas as linhas de A e, para controlar o seu número de iterações, é utilizado o registrador $s4 que terá o valor da posição do primeiro elemento da linha em relação ao endereço de A, ou seja, valerá 0, 12 ou 24
- o loop interno, chamado de Loop_colunas_Bt, percorre todas as colunas de Bt e, para controlar o seu número de iterações, é utilizado o registrador $s5 que terá o valor da posição do primeiro elemento da coluna em relação ao endereço de Bt, ou seja, valerá 0, 4 ou 8
- o loop mais interno, chamado de Loop_mult, percorre todos os elementos da linha de A e da coluna de Bt que estão sendo percorridas. Esses elementos são multiplicados e todos os resultados dessas multiplicações são somados para que seja salvo na matriz resultante da multiplicação (a matriz C).

2.3 Resultados



Esse é o print da tela após a execução do programa.

Dando enfoque na memória:



A matriz resultante é salva a partir do endereço 0x1001006c, com cada elemento da matriz ocupando uma word. Portanto, é possível observar que o resultado da matriz está certa, já que ela deveria ser:

12 10 3

18 -15 4

5 -4 1

e os valores em hexadecimal salvos na memória a partir de 0x1001006c são iguais a matriz acima se ela for lida linha por linha.

3. Questão 2

3.1 Apresentação da questão

A questão 2 solicita um programa em Assembly do MIPS, a ser executado no simulador MARS que realize as mesmas operações da questão 1, porém a matriz resultante do produto A*Bt deveria ser armazenada em um arquivo com extensão .txt. Para isso, foi solicitada a conversão dos elementos da matriz resultante em representações ASCII , a fim de possibilitar sua visualização por uma pessoa no arquivo .txt.

3.2 Estrutura do código

Para a implementação em Assembly das requisições feitas na questão, foram adicionadas ao escopo do código instruções com manipulação de arquivos. Além disso, foi necessário o desenvolvimento e implementação de um algoritmo de conversão dos dados da matriz para a representação ASCII.

3.2.1 Detalhamento do algoritmo de conversão dos dados para ASCII

O algoritmo utilizado para a conversão dos dados da matriz para a representação ascii foi inicialmente projetado em pseudo-linguagem. O pseudo-código pode ser visto na foto a seguir:

pseudocodigo

```
j = la string + (tamanho da string -1) // iterador de j recebe endereço do final do buffer
// pos = 8 x 4 bytes iterador de posição na matriz recebe endereço do último número
// pos -- --> -4
for (int pos = 8; pos >=0; pos --) { // passa por cada número da matriz
    quebra_de_linha = 0
    aux = abs(C[pos]);
    while (aux !=0){ // passa por cada caracter do número

        resto = aux % 10; //resto da divisão (pega o último caracter do número)
        digito = resto +48; // ao somar, corresponde ao número em ASCII
        string[j] = digito; // salva na memória com SB (Store Byte)
        aux = aux/10; // divide por 10 ( diminui o número tirando ultimo caracter já salvo)
        j--;
    }
    if (C[pos]<0){ // se for menor que 0, adiciona na string caracter "-"
        string[j] = 45; // "-" em ASCII
        j--
    }
    string[j] = 32; // adiciona caracter de espaço para separar números
    j--
    quebra_de_linha += 1
    if (quebra_de_linha == 3){
        string[j]= 10 // caracter de quebra de linha
        j--
        quebra_de_linha = 0
    }
}
```

O algoritmo se propõe a realizar a conversão de cada caracter presente em cada número da matriz para o padrão ASCII. (o número “15”, por exemplo, possui o caractere ‘1’ e o caractere ‘5’ em sua representação). Foram consideradas também a necessidade de adicionar o caractere “-” na frente de números negativos , o caractere de espaço entre os números e a quebra de linha para representar a matriz.
Para isso, foi feito o processamento de cada número da matriz, começando do último elemento da matriz até o primeiro (de trás pra frente). O salvamento no Buffer também é feito de trás para frente, da última posição da “string” até a primeira. Isso foi feito para que a ordem de leitura padrão (da esquerda para a direita) fosse garantida.

Alguns detalhes relevantes sobre a implementação da lógica algorítmica do pseudo-código em linguagem Assembly:

- Foram utilizadas estruturas de loop para a implementação das repetições. Esses loops eram limitados e finalizados majoritariamente com a instrução de desvio condicional “branch if not equal”. Enquanto a desigualdade fosse satisfeita, era feito um desvio para o início do loop construído;
- Foi utilizada uma grande variedade de instruções aritméticas estudadas em sala de aula;
- Foi observada a necessidade de utilização da instrução sb “store byte” para que o valor de cada caracter fosse salvo no buffer em tamanho de byte (tamanho padrão da representação de char em ASCII);

Demais detalhes da implementação da lógica em linguagem Assembly para o MIPs podem ser verificados no escopo do código Assembly. O código foi massivamente comentado, para propósitos acadêmicos de aprendizado, a fim de facilitar aos integrantes do grupo o acompanhamento da lógica e da função de cada linha.

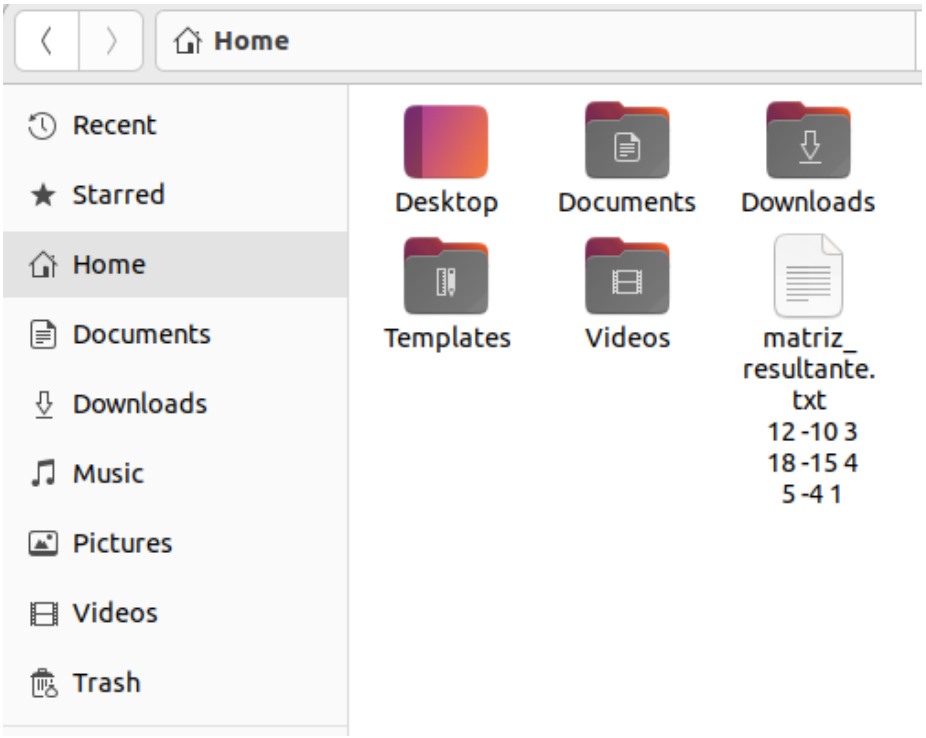
3.2.2 Detalhamento do procedimento de manipulação de arquivo txt:

Para a manipulação com escrita de arquivo, foram utilizadas as chamadas de sistemas apresentadas no slide “MARS: syscall” fornecido no moodle.

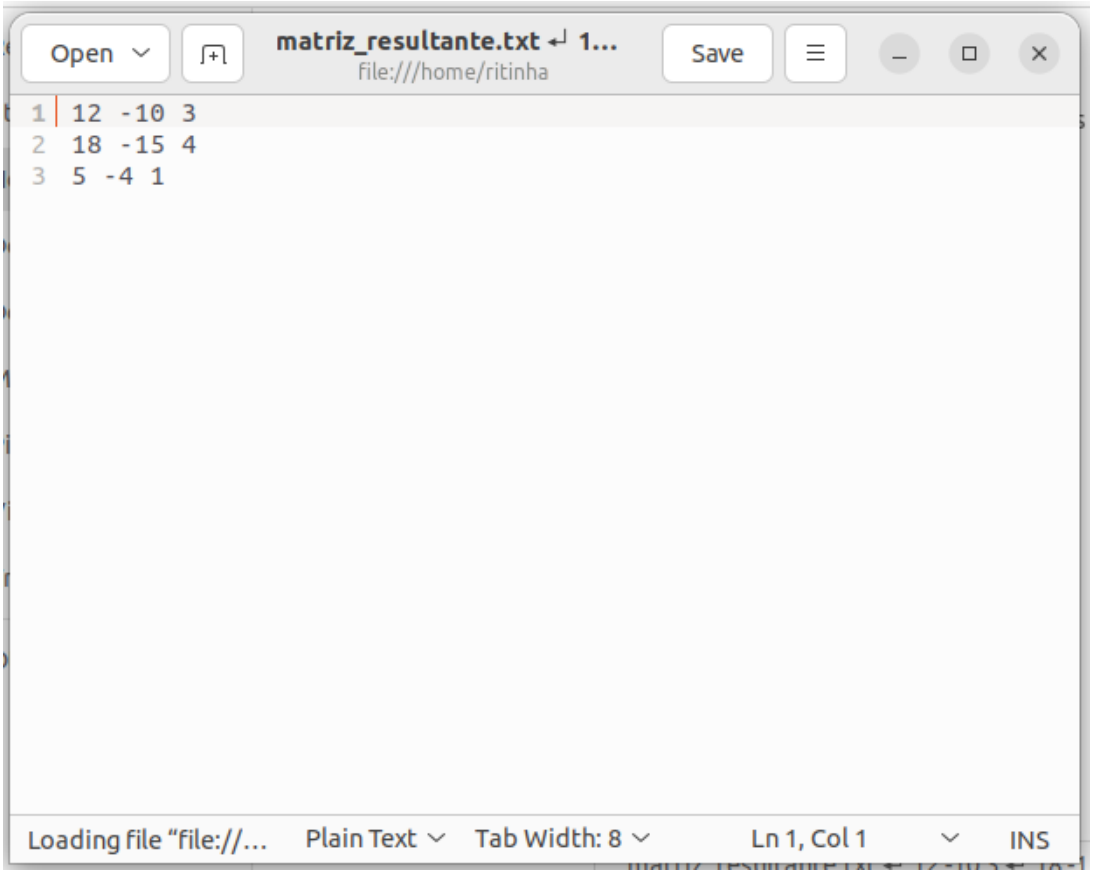
Em resumo, o procedimento adotado para o armazenamento de dados em arquivo foi o seguinte:

- Todos os dados a serem escritos foram armazenados em um buffer na memória (denominado buffer e declarado como .space nas variáveis da memória), com tamanho exato ao necessário para armazenar todos os caracteres ASCII do resultado (neste caso, totalizou-se 27 caracteres).
- Procedimento de abertura do arquivo com chamada de sistema syscall (comando 13)
- Procedimento de escrita em arquivo com chamada de syscall 9 (comando 15), armazenando no arquivo o conteúdo do buffer da memória.
- Procedimento de fechamento do arquivo com chamada syscall (comando 16)

3.3 Resultados



Ao clicar em “Run” na ferramenta MARs, é automaticamente criado no diretório “home” da máquina um arquivo txt com o nome determinado.



Ao abrir o arquivo txt gerado, verifica-se o correto salvamento dos dados da matriz resultante e a correta representação em padrão ASCII, legível pelo ser humano.

● 4. Questão 3

4.1 Apresentação da questão

A questão 3 solicita um programa em Assembly do MIPS, a ser executado no simulador MARS que altere o programa feito na questão 1 para que ele “chame” dois procedimentos: um para multiplicar matrizes (PROC_MUL) e outro para gerar uma matriz transposta (PROC_TRANS).

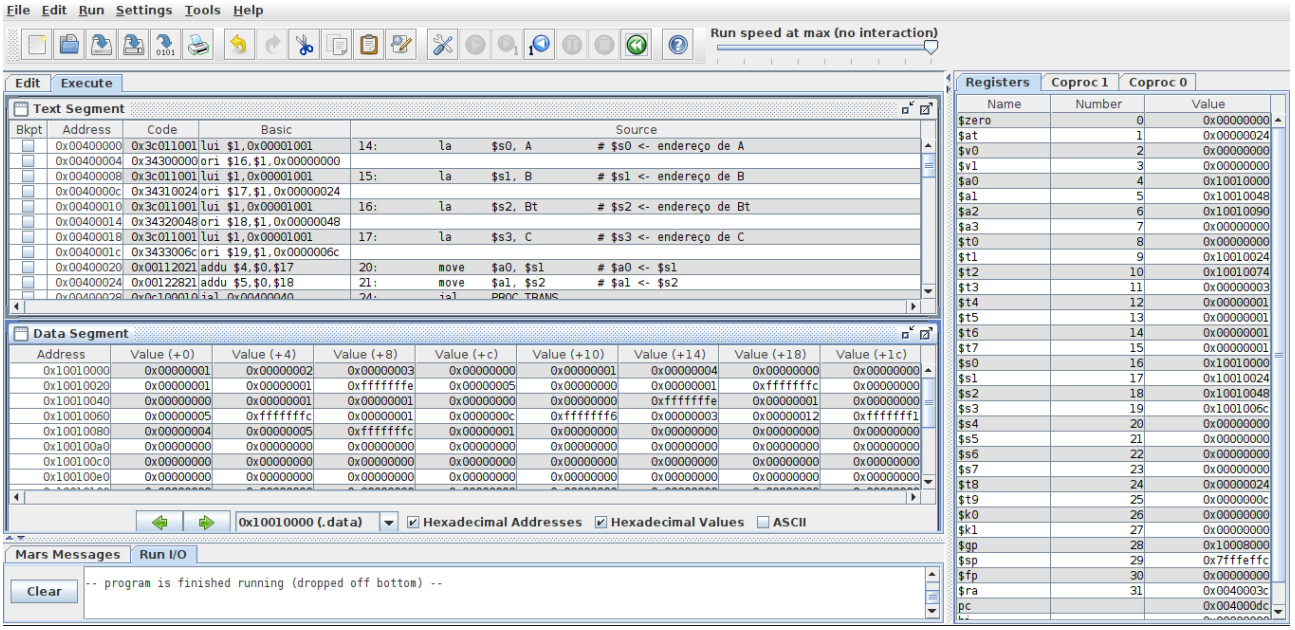
4.2 Estrutura do código

Para a implementação em Assembly das operações requisitadas, foram utilizadas as mesmas instruções usadas na questão 1, além das instruções para chamada de procedimento.

Foi usado o mesmo código da questão 1 com algumas diferenças:

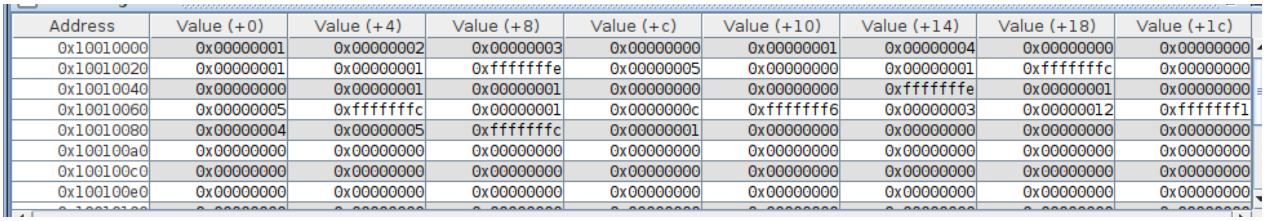
- para o procedimento PROC_TRANS foram utilizados os registradores $a0 e $a1 como argumento do procedimento, os quais são salvos com o endereço inicial de B e de Bt
- para o procedimento PROC_MUL foram utilizados os registradores $a0, $a1 e $a2 como argumento do procedimento, os quais são salvos com o endereço inicial de A, Bt e C

4.3 Resultados



Esse é o print da tela após a execução do programa.

Dando enfoque na memória:



A matriz resultante é salva a partir do endereço 0x1001006c, com cada elemento da matriz ocupando uma word. Portanto, é possível observar que o resultado da matriz está certa, já que ela deveria ser:

12 10 3

18 -15 4

5 -4 1

e os valores em hexadecimal salvos na memória a partir de 0x1001006c são iguais a matriz acima se ela for lida linha por linha.