Linguagem de Montagem e Simulador

Anderson G. S. P. Fantin	9016981
Larissa Fabião da Fonseca	11208367
Lucas Imamura	11208221
Maria Fernanda Basso	11208197
Vinícius Bispo	10875965

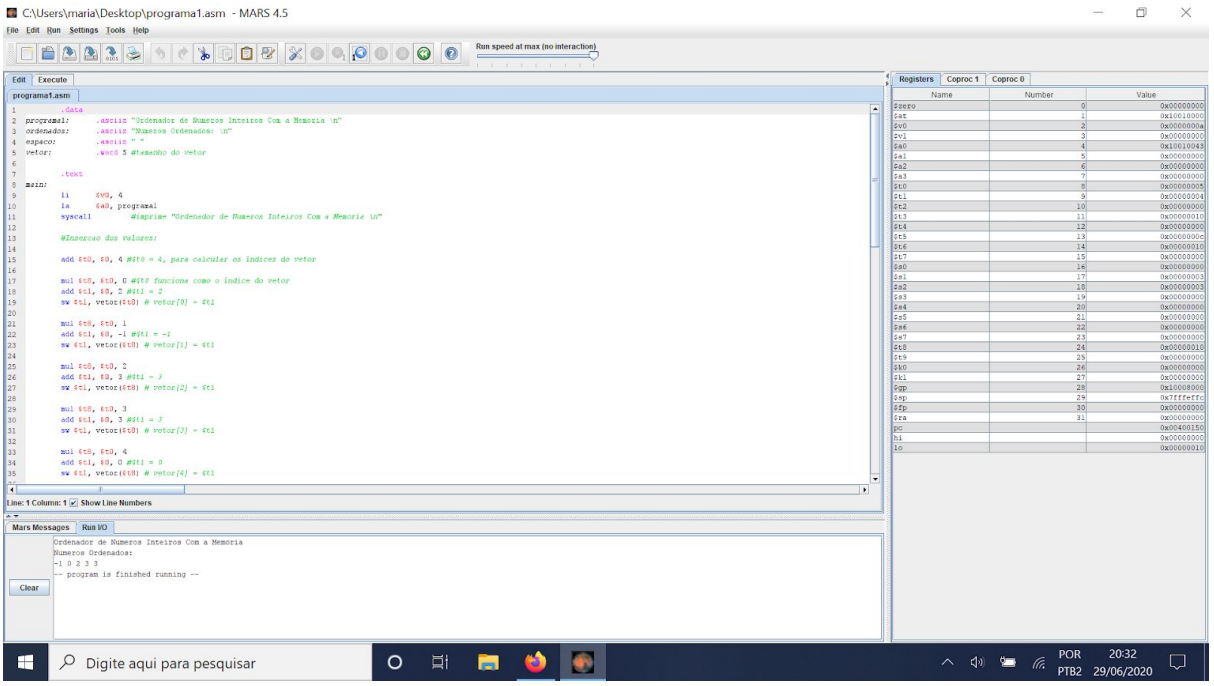
Para essa tarefa realizamos 3 programas que ordenam números inteiros com diferentes entradas. O primeiro programa, com entrada de dados direto da memória, conta com 3 strings em seu bloco .data, as quais são **programa1** que recebe "Ordenador de Numeros Inteiros Com a Memoria \n", **ordenados** que recebe "Numeros Ordenados: \n" e **espaco** que recebe " ". Além disso, em .data também é definido o tamanho do vetor de inteiros, com tamanho 5, sendo esse o limite de quantidade de números que esse algoritmo pode ordenar.

A seguir, temos o bloco .text onde o código efetivamente começa, primeiro imprimimos a primeira string, **programa1**, depois inserimos os valores 2, -1, 3, 3, 0, ou seja, desordenadamente. Após isso, contamos com um código de ordenação que tem 2 estruturas similares ao while em outras linguagens, em cada um deles há um contador para verificar se todo o vetor já foi percorrido, comparando assim, em um if dentro do while interno, se o número na posição relativa ao while externo é maior que todos os seus seguintes a ele.

Dessa forma, caso isso seja verdadeiro, realizamos a troca de posição desses números, caso seja falso “pulamos” para o fim do if, o “doneIf”. Nessa parte do código somamos 1 ao contador do while interno e retornamos para o começo do while interno, quando já percorremos todos os números seguintes a este selecionado no while externo “pulamos” para o fim do while interno, o “doneInterno”; nele somamos 1 ao contador do while externo e retornamos para o começo dele, para assim ele selecionar a posição seguinte do vetor, dessa forma começamos o processo do while interno novamente, até que o while externo tenha percorrido todo o vetor e então “pule” para “doneExterno”.

A partir disso, é impressa a string **ordenados**, e depois imprimimos os números do vetor em sequência intercalados com a string **espaco** (ela imprime um espaço entre os números), para isso realizamos um único while. Por fim, imprimimos a chamada de fim de programa.

Programa 1:

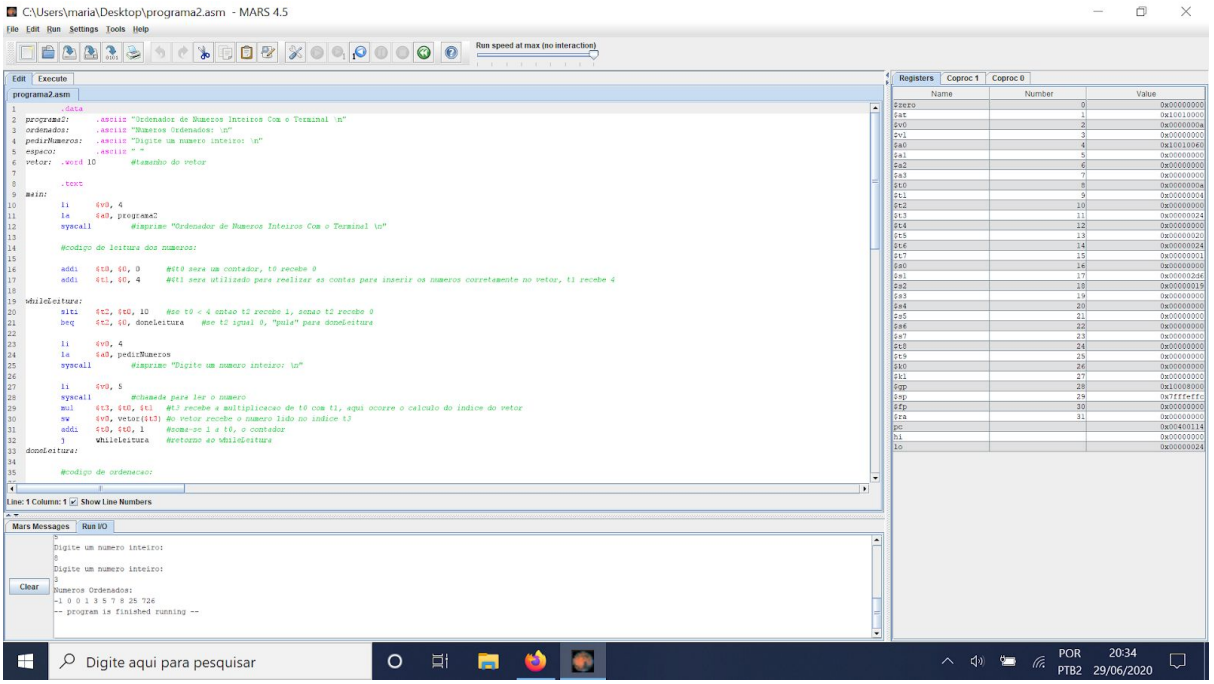


Já o segundo programa, com entrada de dados pelo terminal, conta com 4 strings em seu bloco .data, as quais são **programa2** que recebe "Ordenador de Numeros Inteiros Com o Terminal \n", **ordenados** que recebe "Numeros Ordenados: \n", **pedirNumeros** que recebe "Digite um numero inteiro: \n" e **espaco** que recebe " ". Também, em .data, definimos o tamanho do vetor de inteiros, com tamanho 10, sendo esse o limite de quantidade de números que esse algoritmo pode ordenar.

Logo após isso, se inicia o bloco .text onde imprimimos a string **programa2**, então fazemos um while para leitura, no qual imprimimos a string **pedirNumeros** e então chamamos o sistema para a leitura um número inteiro, por conseguinte o usuário deve digitar um número inteiro, depois esse valor é inserido no vetor (esse while faz 10 iterações).

Como só modificamos a forma de entrada dos números e a quantidade deles, ao final dessa leitura foi utilizado o mesmo código de ordenação e impressão do primeiro programa, mas com mais iterações (no anterior eram 5 e nesse são 10). Finalmente, o algoritmo é encerrado com a chamada de fim de programa.

Programa 2:

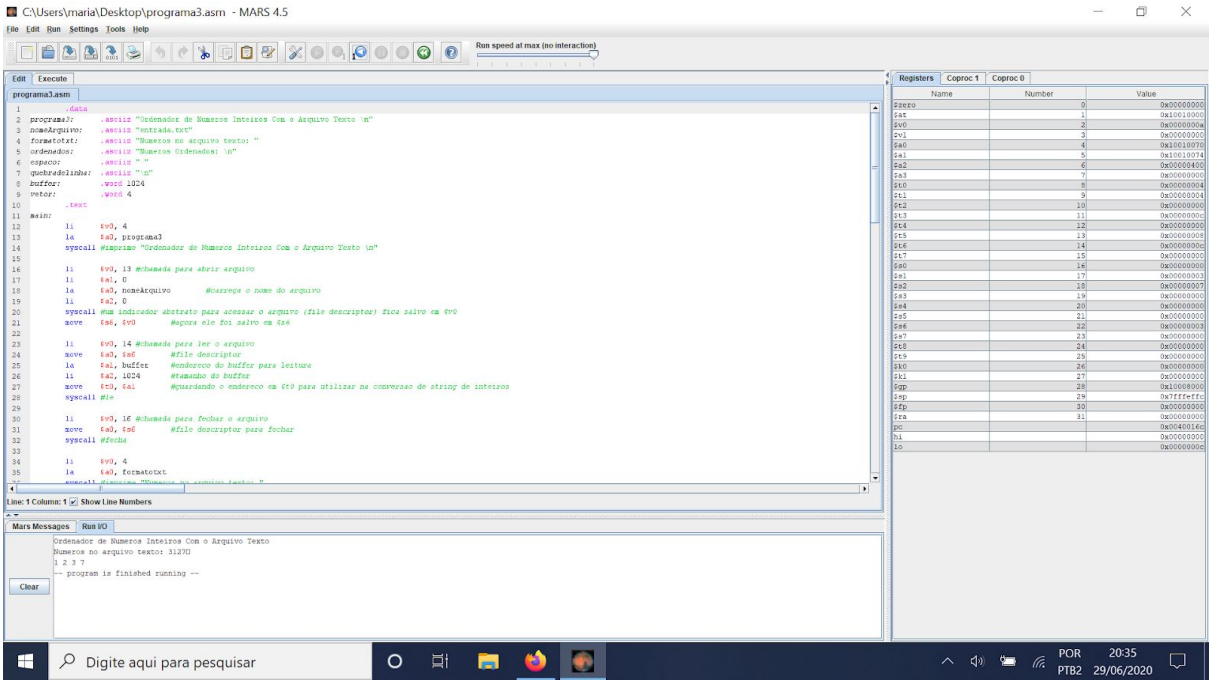


Ainda, a entrada de dados do terceiro programa é através de um arquivo texto, o algoritmo conta com 6 strings em seu bloco .data, as quais são **programa3** que recebe "Ordenador de Numeros Inteiros Com o Arquivo Texto \n", **nomeArquivo** que recebe "entrada.txt", **formatotxt** que recebe "Numeros no arquivo texto: ", **ordenados** que recebe "Numeros Ordenados: \n", **espaco** que recebe **" "** e **quebradelinha** que recebe "\n". Também, em .data, definimos o tamanho do **buffer**, com tamanho 1024, e o tamanho do vetor de inteiros de um dígito não-negativo, com tamanho 4, sendo esse o limite do nosso algoritmo.

Logo após isso, se inicia o bloco .text onde imprimimos a string **programa3**, então abrimos o arquivo que contém somente 4 dígitos e tem seu nome definido na string **nomeArquivo**, depois lemos o arquivo, inserindo seu conteúdo na string **buffer**, também inserimos em $t0 seu endereço, a seguir fechamos o arquivo. A partir desse momento, começamos a conversão, acessamos as posições da string através de $t0, convertendo o carácter em número e inserindo-o no vetor.

Como só modificamos a forma de entrada dos números e a quantidade deles, ao final dessa leitura foi utilizado o mesmo código de ordenação e impressão do primeiro programa, mas com menos iterações (no anterior eram 5 e nesse são 4). Finalmente, o algoritmo é encerrado com a chamada de fim de programa.

Programa 3:


C:\Users\maria\Desktop\programa3.asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
programa3.asm
.data
programa3: .asciiz "Ordenador de Numeros Inteiros Com o Arquivo Texto \n"
nomeArquivo: .asciiz "entrada.txt"
formatotxt: .asciiz "Numeros no arquivo texto: "
ordenados: .asciiz "Numeros Ordenados: \n"
espaco: .asciiz " "
quebradelinha: .asciiz "\n"
buffer: .word 1024
vetor: .word 4
.text
main:
li $v0, 4
la $a0, programa3
syscall #imprime "Ordenador de Numeros Inteiros Com o Arquivo Texto \n"
li $v0, 13 #chamada para abrir arquivo
li $a1, 0
la $a0, nomeArquivo #carrega o nome do arquivo
li $a2, 0
syscall #um indicador abstrato para acessar o arquivo (file descriptor) fica salvo em $v0
move $s6, $v0 #agora ele foi salvo em $s6
li $v0, 14 #chamada para ler o arquivo
move $a0, $s6 #file descriptor
la $a1, buffer #endereco do buffer para leitura
li $a2, 1024 #tamanho do buffer
move $t0, $a1 #guardando o endereco em $t0 para utilizar na conversao de string de inteiros
syscall #le
li $v0, 16 #chamada para fechar o arquivo
move $a0, $s6 #file descriptor para fechar
syscall #fecha
li $v0, 4
la $a0, formatotxt
Line: 1 Column: 1 Show Line Numbers
Mars Messages Run I/O
Ordenador de Numeros Inteiros Com o Arquivo Texto
Numeros no arquivo texto: 3127
1 2 3 7
-- program is finished running --
Clear
Registers Coproc 1 Coproc 0
Name Number Value
$zero 0 0x00000000
$at 1 0x10010000
$v0 2 0x0000000a
$v1 3 0x00000000
$a0 4 0x10010070
$a1 5 0x10010074
$a2 6 0x00000400
$a3 7 0x00000000
$t0 8 0x00000004
$t1 9 0x00000004
$t2 10 0x00000000
$t3 11 0x0000000c
$t4 12 0x00000000
$t5 13 0x00000005
$t6 14 0x0000000c
$t7 15 0x00000000
$s0 16 0x00000000
$s1 17 0x00000003
$s2 18 0x00000007
$s3 19 0x00000000
$s4 20 0x00000000
$s5 21 0x00000000
$s6 22 0x00000003
$s7 23 0x00000000
$t8 24 0x00000000
$t9 25 0x00000000
$k0 26 0x00000000
$k1 27 0x00000000
$gp 28 0x10008000
$sp 29 0x7fffeffc
$fp 30 0x00000000
$ra 31 0x00000000
pc 0x004001
hi 0x00000000
lo 0x0000000c
Digite aqui para pesquisar
POR PTB2 20:35 29/06/2020