UNIVALI – Universidade do Vale do Itajaí

Curso – Ciências da Computação

Disciplina – Arquitetura e Organização de Processadores

Professor – Douglas Rossi de Melo

Alunos – Felipe dos Santos e Nathalia Suzin

PROGRAMAÇÃO EM LINGUAGEM DE MONTAGEM

13/05/2022

Enunciado

Utilizando a linguagem de montagem do MIPS, implemente um procedimento para ordenação de vetores baseado no algoritmo da bolha (bubblesort).

Em C++:

```
#include <iostream>
using namespace std;
int main()
{
    int i, j, n, temp, temp2, var, vet_A[8];
    do{
        cout << "Entre com o número de elementos no vetor (mín=2
e máx=8): ";
        cin >> n;
        if(n<2 or n>8)
            cout << "\nValor inválido.\n";
    }while(n<2 or n>8);
    for(i=0;i<n;i++){
        cout << "\nEntre com o valor Vetor A[" << i << "]: ";
        cin >> vet_A[i];
    }
    for(i=0;i<n-1;i++){
        for(j=0;j<n-1;j++){
           temp=vet_A[j];
           var=j+1;
           if(vet_A[var]<vet_A[j]){
              temp2=vet_A[var];
              vet_A[j]=temp2;
              vet_A[var]=temp;
            }
        }
    }
   cout << endl;
   for(i=0;i<n;i++)
       cout << vet_A[i] << " " << endl;
 }
```

Em MIPS:

```
# Disciplina: Arquitetura e Organização de Processadores
# Atividade: Avaliação 04 – Programação de Procedimentos
# Alunos: Felipe dos Santos e Nathalia Suzin


           .data

Vetor_A:   .word 0,0,0,0,0,0,0,0
sizeVet:   .asciiz    "\nEntre com o tamanho do vetor (mín=2 e
máx=8): "
msgInvalid:     .asciiz "\nValor inválido."
contVet:   .asciiz    "\nVetor_A["
contEnd:   .asciiz    "]: "
testeEnd:  .asciiz "Fim!"

           .text

     li $s1, 7
     li $s2, 8
     li $s3, 9

     li $s6, 0       # i = 0
     li $s7, 0       # j = 0

     j Main


Invalidez:

           li $v0, 4                  # Carrega o serviço 4
     (Ponteiro para string)
           la $a0, msgInvalid         # Carega ptr p/ string
(Mostra a mensagem na tela)
           syscall                    # Chama o serviço 4

# Fim da área que verifica a entrada de dados do tamanho do
vetor


# Início da área que verifica a entrada de dados do tamanho do
vetor
TamanhoVetor:

           li $v0, 4                  # Carrega o servico 4
(Ponteiro para string)
           la $a0, sizeVet            # Carega ptr p/ string
(Mostra a mensagem na tela)
           syscall                    # Chama o servico 4
```

```
        li $v0, 5                       # Carrega o servico 5
        syscall                         # Chama o servico 5
        add $s0, $v0, $zero             # O que o usuario
digitar sera adicionado ao $s0

        blt $s0, 2, Invalidez           # Se o valor digitado
for menor que 2, volta para "Invalidez"
        bgt $s0, 8, Invalidez           # Se o valor digitado
for maior que 8, volta para "Invalidez"

        jr $ra                          # Volta para Main se
tamanho é válido!

# Área para leitura dos vetores

VoltarMain:
     subi $sp, $sp, 16     # Cria uma pilha com 1 espaço
     sw $ra, 0($sp)        # Guarda valor do $ra (main)
     sw $a0, 4($sp)        # Guarda o tamanho do vetor na
pilha
     sw $s1, 8($sp)        # Carrega o valor inicial de $s1
na pilha
     sw $s2, 12($sp)       # Carrega o valor inicial de $s2
na pilha
     sw $s3, 16($sp)       # Carrega o valor inicial de $s3
na pilha

LerVetor:

     li $v0, 4             # Carrega o servico 4 (Ponteiro
para string)
        la $a0, contVet    # Carrega ptr p/ string (Mostra
a mensagem na tela)
        syscall            # Chama o servico 4

        li $v0, 1          # Carrega o servico 1 (inteiro)
        la $a0, ($s6)      # Carrega no $a0 o valor inteiro
do indice
        syscall            # Chama o servico 1

        li $v0, 4          # Carrega o servico 4 (Ponteiro
para string)
        la $a0, contEnd    # Carrega ptr p/ string (Mostra
a mensagem na tela)
        syscall            # Chama o servico 4

     li $v0, 5                          # Carrega o servico 5
        syscall                         # Chama o servico 5
        add $s1, $v0, $zero             # O que o usuario
digitar sera adicionado ao $s1

        jal  MulIndice                  # Função que faz a
multiplicação do índice
```

```
          add $t7, $t7, $a1          # Guarda no $t7, End.
Absoluto = 4*i + End.Base
          sw $s1, 0($t7)             # Salva no vetor o número
que o usuario digitou, usando o End. Absoluto


          addi $s6, $s6, 1           # i = i + 1

          blt $s6, $s0, LerVetor     # Se indice for menor que
o valor escolhido pelo usuario, volta para LerVetores

          addi $s6, $zero, 0         # i = 0
          lw $a0, 0($sp)             # Carrega o tamanho do
vetor de volta para $a0

          subi $t3, $s0, 1           # Tamanho do vetor - 2,
para ser usado no BubbleSort


# BubbleSort vai realizar a tarefa de arruamar os elementos no
vetor

BubbleSort:
          beq $s7, $s0, finalizar

          jal MulIndice              # 4*i
          add $s3, $t7, $a1          # Guarda em $s3 o End.
Absoluto do Vetor_A [i]
          lw $s1, 0($s3)             # Guarda o Valor 1

          addi $s6, $s6, 1# i = i + 1

          jal MulIndice              # 4*i
          add $s4, $t7, $a1          # Guarda em $s4 o End.
Absoluto do Vetor_A [i+1]
          lw $s2, 0($s4)             # Guarda o Valor 2

          bgt $s1, $s2, troca        # Se o valor 1 > valor 2,
volta para o BubbleSort
          jal ntroca


# Calcular 4*i
MulIndice:
          mul $t7, $s6, 4            # Guarda no $s7 o valor
do indice multiplicado por 4
          jr $ra                     # Retorna para Calcular o
endereço absoluto e guardar no vetor
# Fim calculo 4*i
```

```
troca:
          sw $s2, 0($s3)              # Guarda o valor do
Vetor_A [i + 1] na posicao  Vetor_A [i]
          sw $s1, 0($s4)              # Guarda o valor do
Vetor_A [i] na posicao  Vetor_A [i + 1]


          blt $s6, $t3, BubbleSort    # Enquanto i < tamanho
- 1
          addi $s7, $s7, 1            # j = j + 1
          li $s6, 0
          blt $s7, $s0, BubbleSort    # Enquanto j < tamanho

          j finalizar

          jr $ra                      # Volta para a main
apos o comando = " jal VoltarMain "

ntroca:

          blt $s6, $t3, BubbleSort    # Enquanto i < tamanho
- 1
          addi $s7, $s7, 1            # j = j + 1
          li $s6, 0
          blt $s7, $s0, BubbleSort    # Enquanto j < tamanho

          j finalizar

finalizar:

          lw $s3, 16($sp)             # Carrega o tamanho do
vetor em $s3
          lw $s2, 12($sp)             # Carrega o tamanho do
vetor em $s2
          lw $s1, 8($sp)              # Carrega o tamanho do
vetor em $s1
          lw $a0, 4($sp)              # Carrega o tamanho do
vetor em $a0
          lw $ra, 0($sp)              # Carrega o endereco
$ra (main) de volta em $ra
          addi $sp, $sp, 16           # Desforma a pilha

          jr $ra
# Fim do BubbleSort

MostrarTela:                          # Mostra o resultado do
vetor na tela

          li $v0, 4                   # Carrega o servico 4
(Ponteiro para string)
          la $a0, contVet             # Carrega ptr p/ string
(Mostra a mensagem na tela)
          syscall
```

```
          li $v0, 1                    # Carrega o servico 1
(inteiro)
          la $a0, ($s6)                # Carrega no $a0 o
valor inteiro do indice
          syscall                      # Chama o servico 1

          li $v0, 4                    # Carrega o servico 4
(Ponteiro para string)
          la $a0, contEnd              # Carrega ptr p/ string
(Mostra a mensagem na tela)
          syscall                      # Chama o servico 4

          mul $t7, $s6, 4              # $t7 = 4*1
          add $t7, $a1, $t7            # $t7 = endereco base +
4*i
          lw $s7, 0($t7)               # $t2 receber o valor
de A[i]

          li $v0, 1                    # Carrega o servico 1
(inteiro)
          la $a0, ($s7)                # Carrega no $a0 o
valor inteiro do indice
          syscall                      # Chama o servico 1

          addi $s6, $s6, 1             # i = i + 1
          blt $s6, $s0, MostrarTela    # Enquanto i < N de
vetores
          jr $ra                       # Volta para Main

Main:
          la $a1, Vetor_A              # Carrega no $a1 o
endereço do Vetor_A

          jal TamanhoVetor             # Manda para leitura do
tamanho do Vetor
     add $a0, $s0, $zero               # Guarda no $s0 o
tamanho do vetor

          jal VoltarMain               # Manda para a pilha
que irá guaradar o $a0 e o $ra de retorno para proxima linha
do main

          li $s6, 0                    # i = 0
          jal MostrarTela              # Funcao para mostrar
os vetores de forma ordenada na tela

Exit:
     nop                               # NOP  - Sem mais
operacoes, parada do programa
```

Relatório

Inicialmente, em .data, são declaradas as mensagens que serão chamadas na tela pelo syscall e, também, declarado o tamanho do vetor utilizado no código. Neste caso, o vetor (Vetor_A) terá um tamanho máximo de 8 elementos, todos inicializados com 0.

```
            .data

Vetor_A:          .word 0,0,0,0,0,0,0,0
sizeVet:          .asciiz    "\nEntre com o tamanho do vetor
(mín=2 e máx=8): "
msgInvalid:       .asciiz "\nValor inválido."
contVet:          .asciiz    "\nVetor_A["
contEnd:          .asciiz    "]: "
testeEnd:         .asciiz "Fim!"
```

As primeiras instruções, iniciando o programa em .text, guardarão os valores 7, 8 e 9 e o índice em registradores salvos, pois o índice será utilizado até o fim do programa e os três valores citados servirão de teste para o funcionamento da pilha.

```
          .text

      li $s1, 7
      li $s2, 8
      li $s3, 9

      li $s6, 0
      li $s7, 0

      j Main
```

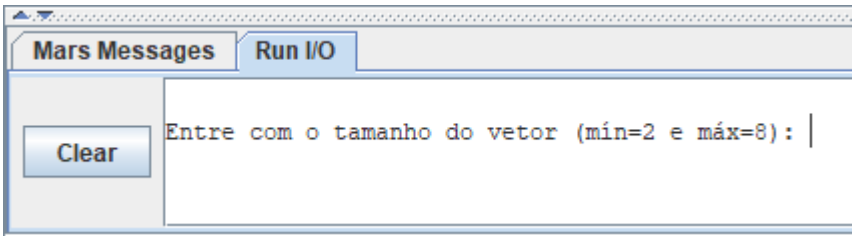
Após o armazenamento aos registradores, o programa executará um jump para a função principal, Main. Já na função, em sua primeira instrução, é feita a cópia do endereço do Vetor_A para $a1 e em seguida um jal (jump and link) para o campo denominado TamanhoVetor.

```
Main:
      la $a1, Vetor_A
      jal TamanhoVetor
      add $a0, $s0, $zero
      jal VoltarMain
      li $s6, 0
      jal MostrarTela

      j Exit
```

No TamanhoVetor, será apresentada uma mensagem e, após, a leitura do valor que o usuário digitar no console. Este valor é correspondente a quantidade de elementos que o indivíduo deseja que o vetore possua, além disso, o número que o usuário digitar deve estar entre o intervalo de 2 e 8, pois 8 é a quantidade máxima de elementos que os vetores podem obter e é necessário que tenha pelo menos 2 elementos para que possa ser feita a comparação e a ordenação.

```
TamanhoVetor:

            li $v0, 4
            la $a0, sizeVet
            syscall

            li $v0, 5
            syscall
            add $s0, $v0, $zero

            blt $s0, 2, Invalidez
            bgt $s0, 8, Invalidez

            jr $ra
```

O código 4 (impressão de frase) é carregado e será apresentada a mensagem guardada em *sizeVet* na tela:



Endereço: 0x0040002c

Registrador	*Número*	*Valor*
$v0	2	0x00000004

O código 5 (leitura de valor inteiro) é carregado e o número que o usuário digitar será armazenado no registrador $s0.

```
add $s0, $v0, $zero
```

Endereço: 0x00400040

Registrador	*Número*	*Valor*
$s0	16	0x00000002

Mas, antes de prosseguir com o restante do código, serão feitas as validações, denominadas por:

- be less than (blt – se menor que) → onde se o número digitado for menor que 2, então vá para o destino informado.
  - *Endereço: 0x00400044*
- be greater than (bgt – se maior que) → onde se o número digitado for maior que 8, então vá para o destino informado.
  - *Endereço: 0x0040004c*

Assim como descrito no programa, se o valor não for validado terá um desvio para o espaço denominado Invalidez. Nele, é carregado o código 4 (impressão de frase) e feita uma chamada do syscall, que apresentará a mensagem "Valor inválido", declarada em msgInvalid, na tela e em seguida cairá no Main novamente e será refeita a pergunta até que o valor inserido seja válido, como mostrado nas imagens abaixo.

```
Invalidez:
          li $v0, 4
          la $a0, msgInvalid
          syscall
```

O código 4 (impressão de frase) é carregado e será apresentada a mensagem guardada em *msgInvalid* na tela:

Endereço: 0x0040001c

Registrador	*Número*	*Valor*
$a0	4	0x10010050

Feito isto e o valor inserido validado, o programa prossegue executando um jr $ra para o endereço salvo anteriormente no Main, que logo opera uma instrução que copia a quantidade de elementos para $a0.

Endereço: 0x004001d4

Registrador	*Número*	*Valor*
$a0	4	0x00000005

Então, é trazido o segundo jal ao VoltarMain, guardando o endereço da instrução seguinte para depois retornar. O VoltarMain aciona a pilha, alocando 5 espaços na memória.

```
VoltarMain:
      subi $sp, $sp, 16
      sw $ra, 0($sp)
      sw $a0, 4($sp)
      sw $s1, 8($sp)
      sw $s2, 12($sp)
      sw $s3, 16($sp)
```

Endereço: 0x0040005c

Registrador	*Número*	*Valor*
$sp	29	0x7fffefec

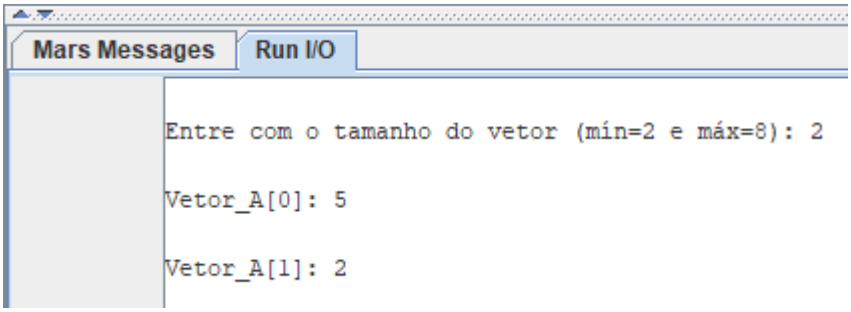
Após o término do campo VoltarMain o código cairá para o laço LerVetor. Este laço é responsável por coletar os dados do vetor_A, ou seja, pedirá e guardará os valores do tipo inteiro inseridos em seus respectivos índices até o término das inserções, respeitando a quantidade de elementos determinada pelo usuário anteriormente, no campo TamanhoVetor.

```
LerVetor:

            li $v0, 4
            la $a0, contVet
            syscall

            li $v0, 1
            la $a0, ($s6)
            syscall

            li $v0, 4
            la $a0, contEnd
            syscall

            li $v0, 5
            syscall
            add $s1, $v0, $zero
            jal MulIndice

            add $t7, $t7, $a1
            sw $s1, 0($t7)

            addi $s6, $s6, 1
            blt $s6, $s0, LerVetor

            addi $s6, $zero, 0
            lw $a0, 0($sp)

            subi $t3, $s0, 1
```

O código 4 (impressão de frase) é carregado e a mensagem é enviada ao console. Observa-se que há uma chamada do índice, carregado pelo código 1 (impressão de número inteiro) e em seguida outra chamada do código 4, a qual fecha a mensagem que foi enviada. Foi feito assim para dar o efeito de contagem do índice, pois ao final, em “addi $s6, $s6, 1” é feito o acréscimo de 1 ao $s6 (registrador do índice), assim como um contador. Consequentemente, é feita a leitura dos números que o usuário impor, carregados pelo código 5 (leitura de número inteiro). Assim, o LerVetor

continuará sendo executado até que o limite de elementos declarado pelo usuário chegue ao limite, sendo validado pelo blt (be less than – se menor que).

Exemplo:



O código 4 (impressão de frase) é carregado e será apresentada a mensagem guardada em *contVet* na tela:

Endereço: 0x0040007c

Registrador	*Número*	*Valor*
$a0	4	0x10010061

O código 1 (impressão de inteiro) é carregado e será apresentado o índice:

Endereço: 0x0040008c

Registrador	*Número*	*Valor*
$a0	4	0x00000000

O código 4 (impressão de frase) é carregado e será apresentada a mensagem guardada em *contEnd* na tela:

Endereço: 0x00400098

Registrador	*Número*	*Valor*
$a0	4	0x1001006b

O código 5 (leitura de valor inteiro) é carregado e o número que o usuário digitar:

Endereço: 0x004000ac

Registrador	*Número*	*Valor*
$s1	17	0x00000003

Zerando o contador para sua próxima utilização, a instrução que carrega o tamanho do vetor de volta para $s0 é acionada em seguida e o último comando do campo é subtrair 2 do tamanho do vetor para que assim seja usado no BubbleSort, o campo abaixo.

```
BubbleSort:
      beq $s7, $s0, finalizar

      jal MulIndice
      add $s3, $t7, $a1
      lw $s1, 0($s3)

      addi $s6, $s6, 1

      jal MulIndice
      add $s4, $t7, $a1
      lw $s2, 0($s4)

      bgt $s1, $s2, troca
      jal ntroca
```

Em BubbleSort, é feita a ordenação dos valores do vetor, colocando-os em ordem crescente. Esta parte do código é iniciada com uma validação, o beq.

- be equal (beq – se igual então) → onde se o índice for igual ao elemento da vez então vá para o campo determinado. ● *Endereço: 0x004000d8*

Logo em seguida é executado um jal para MulIndice, campo responsável pela multiplicação do índice, ou seja, o cálculo do endereço absoluto.

```
MulIndice:
      mul $t7, $s6, 4
      jr $ra
```

Endereço: 0x004000b0

Registrador	*Número*	*Valor*
$ra	31	0x004000b4

Voltando, o endereço absoluto é salvo em $s3 e armazenando o primeiro valor pelo load word. Assim, é feito o acréscimo de 1 ao índice para seguir com a contagem e mais um jal para MulIndice para seu cálculo como anteriormente, mas desta vez guardando-o em $s4 e armazenando o segundo valor.

- be greater than (bgt – se maior que) → onde se o valor em 1 for maior que o valor em 2, então vá para o destino informado. ●*Endereço: 0x004000f8*

Caso o valor em $s1 for maior que o valor em $s2, então o código será encaminhado para o campo ‘troca’, senão executará um jal para ‘ntroca’.

1. CASO 1º VALOR MAIOR QUE 2º VALOR

```
troca:

      sw $s2, 0($s3)
      sw $s1, 0($s4)

      blt $s6, $t3, BubbleSort
      addi $s7, $s7, 1
      li $s6, 0
      blt $s7, $s0, BubbleSort

      j finalizar

      jr $ra
```

A primeira instrução guarda o valor do Vetor_A [i + 1] na posição Vetor_A [i] seguida de uma instrução que guarda o valor do Vetor_A [i] na posicao Vetor_A [i + 1]. Logo, são feitas as validações por blt. Neste caso, ele voltará para o BubbleSort enquanto i for menor que o tamanho do vetor menos 1. Se for maior, será feito o acréscimo de 1 ao contador e uma segunda validação por blt, onde ele voltará para o BubbleSort enquanto j for menor que o tamanho do vetor. Finalizando com um jump para o campo 'finalizar'.

2. CASO 2º VALOR MAIOR QUE 1º VALOR

```
ntroca:


      blt $s6, $t3, BubbleSort
      addi $s7, $s7, 1
      li $s6, 0
      blt $s7, $s0, BubbleSort

      j finalizar
```

A primeira instrução é de uma validação por blt, onde ele voltará para o BubbleSort enquanto i for menor que o tamanho do vetor menos 1. Se for maior, será feito o acréscimo de 1 ao contador e uma segunda validação por blt, onde ele voltará para o BubbleSort enquanto j for menor que o tamanho do vetor. Finalizando com um jump para o campo 'finalizar'.

Já o campo Finalizar é responsável por desempilhar o que foi empilhado no ínicio do programa, trazendo de volta as informações necessárias para cumprir as solicitações do usuário.

```
finalizar:

      lw $s3, 16($sp)
      lw $s2, 12($sp)
      lw $s1, 8($sp)
      lw $a0, 4($sp)
      lw $ra, 0($sp)
      addi $sp, $sp, 16

      jr $ra
```

Endereço: 0x00400154

Registrador	*Número*	*Valor*
$s3	19	0x00000009

Endereço: 0x00400158

Registrador	*Número*	*Valor*
$s2	18	0x00000008

Endereço: 0x0040015C

Registrador	*Número*	*Valor*
$s1	17	0x00000007

Endereço: 0x00400160

Registrador	*Número*	*Valor*
$a0	4	0x00000002

Endereço: 0x00400164

Registrador	*Número*	*Valor*
$ra	31	0x004001dc

Endereço: 0x00400168

Registrador	*Número*	*Valor*
$sp	29	0x7fffeffc

Finalizando o campo anterior, é feito o retorno para a funçao principal, onde o índice é reiniciado e o programa executa seu último jal (jump and link) para o campo MostrarTela, responsável por mostrar o vetor já ordenado. Este campo repetirá até que todos os elementos sejam mostrados no console de interface, com o auxílio da instrução de validação por blt.

```
MostrarTela:

            li $v0, 4
            la $a0, contVet
            syscall

            li $v0, 1
            la $a0, ($s6)
            syscall

            li $v0, 4
            la $a0, contEnd
            syscall

            mul $t7, $s6, 4
            add $t7, $a1, $t7
            lw $s7, 0($t7)

            li $v0, 1
            la $a0, ($s7)
            syscall

            addi $s6, $s6, 1

            blt $s6, $s0, MostrarTela

            jr $ra
```

O código 4 (impressão de frase) é carregado e será apresentada a mensagem guardada em *contVet* na tela:

Endereço: 0x00400174

Registrador	*Número*	*Valor*
$a0	4	0x10010061

O código 1 (impressão de número inteiro) é carregado e o índice é mostrado:

Endereço: 0x00400184

Registrador	*Número*	*Valor*
$a0	4	0x00000000

O código 4 (impressão de frase) é carregado e será apresentada a mensagem guardada em *contEnd* na tela:

Endereço: 0x00400190

Registrador	Número	Valor
$a0	4	0x1001006b

O código 1 (impressão de número inteiro) é carregado e o valor 2 é mostrado:

Endereço: 0x004001b0

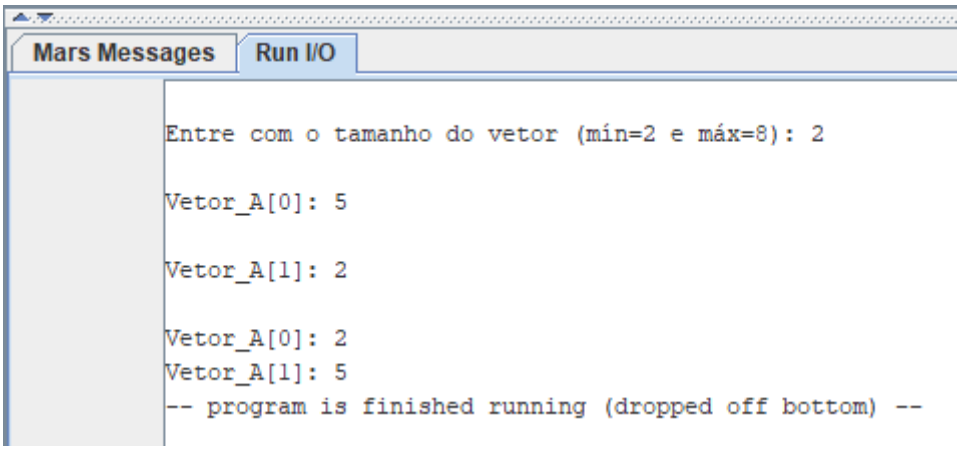
Registrador	Número	Valor
$s7	23	0x00000002
$a0	4	0x00000002

O código 1 (impressão de número inteiro) é carregado e o valor 5 é mostrado:

Endereço: 0x004001b0

Registrador	Número	Valor
$s7	23	0x00000005
$a0	4	0x00000005

Ao final, o console terminará com as seguintes mensagens na tela:



E enfim, para finalizar o programa, o código encerrará no campo Exit, apesentando nenhuma instrução a mais.

```
Exit:
      nop
```

Endereço: 0x004001e4