UNIVERSIDADE DO VALE DO ITAJAÍ
CAMPUS KOBRASOL – SÃO JOSÉ
CIÊNCIA DA COMPUTAÇÃO

Arquitetura de Computadores

Avaliação 03 – Programação de Procedimentos

Professor: Douglas Rossi de Melo
Aluno: Matheus Henrique Schaly

Data de entrega: 06/10/2017

Resumo

Neste trabalho desenvolvemos um programa em linguagem de montagem (assembly), utilizando a arquitetura MIPS e o programa MARS versão 4.5. O programa é capaz de encontrar o máximo divisor comum entre dois números inteiros.
Capturas de tela foram utilizadas para demonstrar mais facilmente a execução do programa.

Sumário

Introdução

Este trabalho tem como finalidade expor a demonstração de um programa implementado em linguagem de montagem (assembly) por meio da arquitetura MIPS.

Tal programa é capaz de ler dois números inteiros oriundos da entrada do usuário, e encontrar o máximo divisor comum dentre eles.

Algumas das funções do programa incluem: impressão de mensagens, leitura de inteiros, um loop, desvios condicionais e principalmente o uso do registrador sp (stack pointer) e a chamada a um procedimento.

I – Código assembly MIPS

1 – Código

Código completo do programa (dentro dos retângulos), seguido por comentários sobre o código destacado. Sentenças com fonte reduzidas (dentro do código) são comentários.

```
#Disciplina: Arquitetura e Organização de Computadores
#Atividade: Avaliação 03 – Programação de Procedimentos
#Grupo: Matheus Henrique Schaly
```

Cabeçalho.

```
.data
     #Creates RAM variables
     message1: .asciiz        "Enter first number: "
     message2: .asciiz        "Enter second number: "
```

Variáveis armazenadas na memória RAM, message1 e message2 serão utilizadas na interface com o usuário (para a entrada de dados).

```
.text
     #Prints a text
     li   $v0, 4                    #Command to print a text
     la   $a0, message1             #Load address of mensagem1 to a0
     syscall                        #Do it

     #Reads integer
     li   $v0, 5                    #Read an integer and store it in v0
(first number)
     syscall                        #Do it

     #Stores integer
     move $s0, $v0                  #Move to s0 (first number) the integer
in v0 (first number)

     #Prints a text
     li   $v0, 4                    #Command to print a text
     la   $a0, message2             #Load address of mensagem2 to a0
     syscall                        #Do it

     #Reads integer
     li   $v0, 5                    #Read an integer and store it in v0
(second number)
     syscall                        #Do it

     #Stores integer
     move $s1, $v0                  #Move to s1 (second number) the integer
in v0 (second number)
```

Imprime a message1, requisitando um número ao usuário. Em seguida lê e armazena o número fornecido pelo usuário. Tal processo é repetido a fim de receber os dois número que serão utilizados para calcular o máximo divisor comum.

```
      #Stores integer in argument registers
      move $a0, $s0                    #Move to a0 (first number) the integer
in s0 (first number)
      move $a1, $s1                    #Move to a1 (second number) the integer
in s1 (second number)
```

Armazena as variáveis lidas nos registradores de argumento, para em seguida entrar no procedimento.

```
      #Enters the procedure (gcd)
      jal  gcd                         #jal (jump and link). Set ra to PC
(return address) then jump to gcd (greatest common divisor). $ra <- PC + 4
```

Chama o procedimento.

```
      #Moves v0 (gcd answer) to s2 (asnwer)
      move $s2, $v0                    #Move to s2 (answer) the integer in v0
(gcd answer)

      #Prints s2 (answer)
      li   $v0,      1                       #Command to print a integer
      move      $a0,      $s2                      #Move s2 (answer) to a0
      syscall                          #Do it

      #Exits the program
      j    exitProgram                       #Jump to exitProgram
```

Tal código será executado após a execução do procedimento. Move o resultado do procedimento para um registrador e imprime tal registrador, que possui a resposta final do programa (máximo divisor comum). E em seguida, pula para exitProgram que encerra o programa.

```
      #The procedure gcd (greatest common divisor)
gcd:

      #Stacks values to be immutably returned
      addi $sp, $sp, -4                #Add sp (unchanged stack pointer) and -
4 (bytes to be moved from sp) and put it back into sp (stack pointer moved by -4
bytes)
      sw   $s0, ($sp)                        #Store word from s0 (first number)
in sp (stack pointer moved by -4 bytes)
      addi $sp, $sp, -4                #Add  sp  (stack  pointer  moved  by  -4
bytes) and -4 (bytes to be moved from sp) and put it back into sp (stack pointer
moved by -8 bytes)
      sw   $s1, ($sp)                        #Store   word   from   s1   (second
number) in sp (stack pointer moved by -8 bytes)
```

Empilha os valores originais fornecidos pelo usuário na pilha (referenciado pelo sp, stack pointer). Tal empilhamento visa a imutabilidade dos valores fornecidos pelo usuário.

```
while:
      #While condition to check if s0 (first number) is equal to s1 (second number)
      beq  $s0, $s1, gcdExit              #Branch to gcdExit if s0 (first
number) is equal to s1 (second number)

      #If condition to check if s0 (first number) is less than s1 (second number)
      blt  $s0, $s1, if                   #Branch to if if s0 (first number) is
less than s1 (second number)

      #Else condition, s0 (first number) is greater than s1 (second number)
      sub  $s0, $s0, $s1                  #Subtract s0 (first number) from s1
(second number) and put it back in s0 (modified first number)
      j    while                          #Jump to while loop

      #If condition, s0 (first number) is less than s1 (second number)
if:

      #If condition body s0 (first number) is less than s1 (second number)
      sub  $s1, $s1, $s0                  #Subtract s1 (second number) from s0
(first number) and put it back in s1 (modified second number)
      j    while                          #Jump to while loop

      #While end condition, s0 (first number) is equal to s1 (second number)
gcdExit:
```

Executa o algoritmo que encontrará o máximo divisor comum dos dois número inteiros.

```
      #Move the answer (s0 (first number)) to v0 (return register)
      move $v0, $s0                       #Move to v0 (return value) the integer
in s0 (first number)
```

Move o resultado para o registrador de retorno.

```
      #Unstacks immutably returned values
      lw   $s1, ($sp)                     #Load word from s1 (second number)
from sp (stack pointer moved by -8 bytes)
      addi $sp, $sp, 4                    #Add sp (stack pointer moved by -8
bytes) and 4 (bytes to be moved from sp) and put it back into sp (stack pointer
moved by -4 bytes)
      lw   $s0, ($sp)                           #Store word from s0 (first
number) in sp (stack pointer moved by -4 bytes)
      addi $sp, $sp, 4                    #Add sp (stack pointer moved by -4
bytes) and 4 (bytes to be moved from sp) and put it back into sp (unchanged stack
pointer)
```

Desempilha os valores da pilha (através do stack pointer), retornando seus valores originais.

```
#Returns to main
      jr    $ra                          #jr  (jump  register  unconditionally).
Jump to statement whose address is in ra (old main PC address). PC <- $ra
```

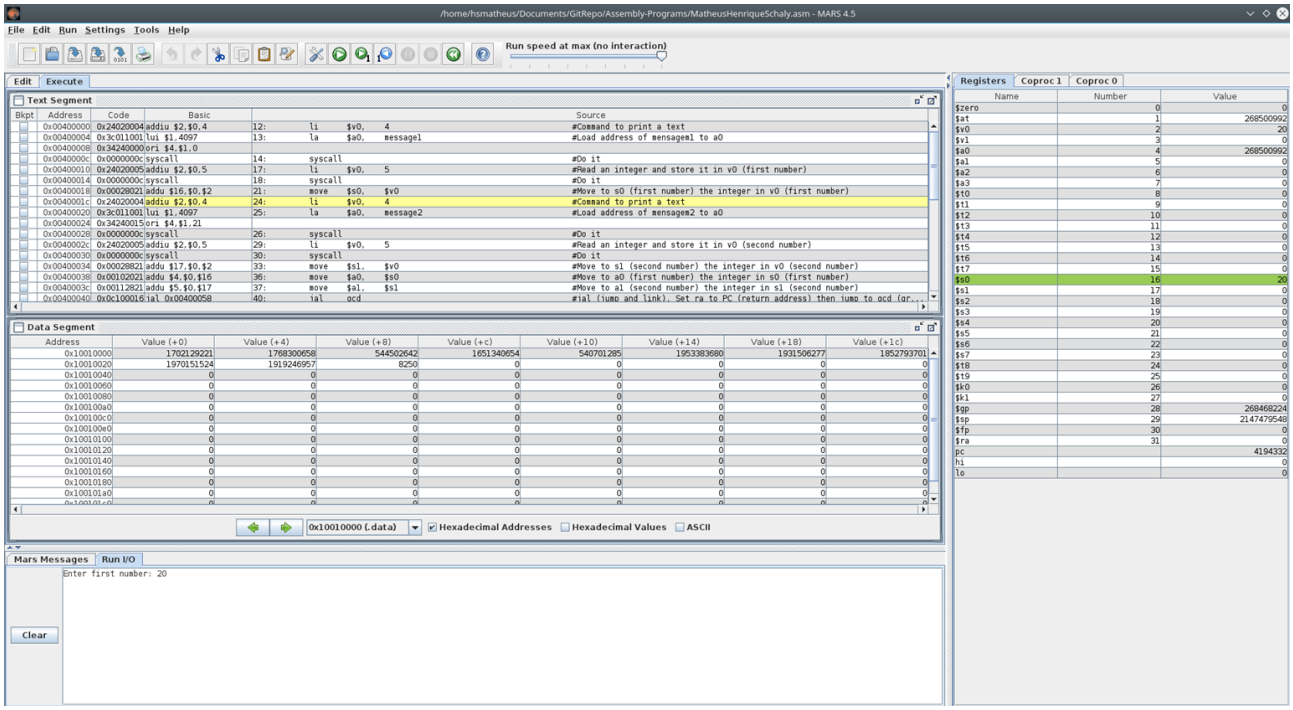
Encerra o procedimento, retornando ao main.

```
exitProgram:
```

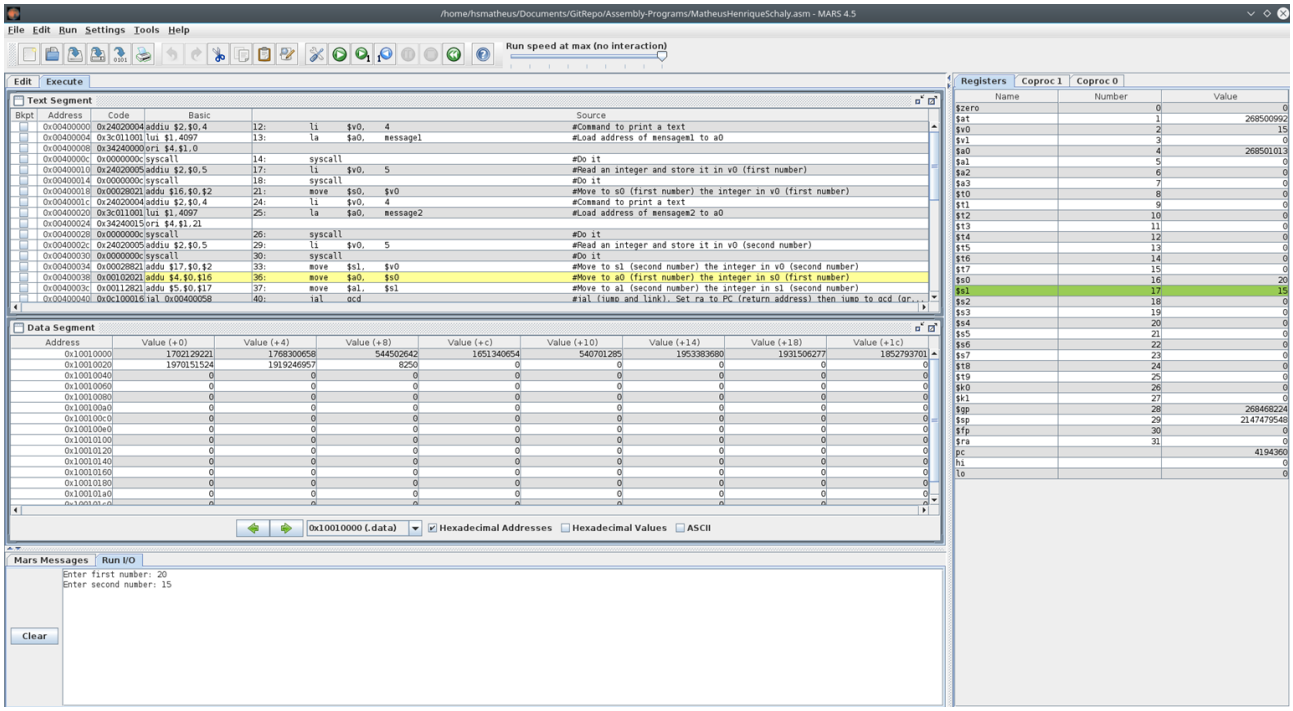
II – Resultados e análises

1 – Análise dos resultados

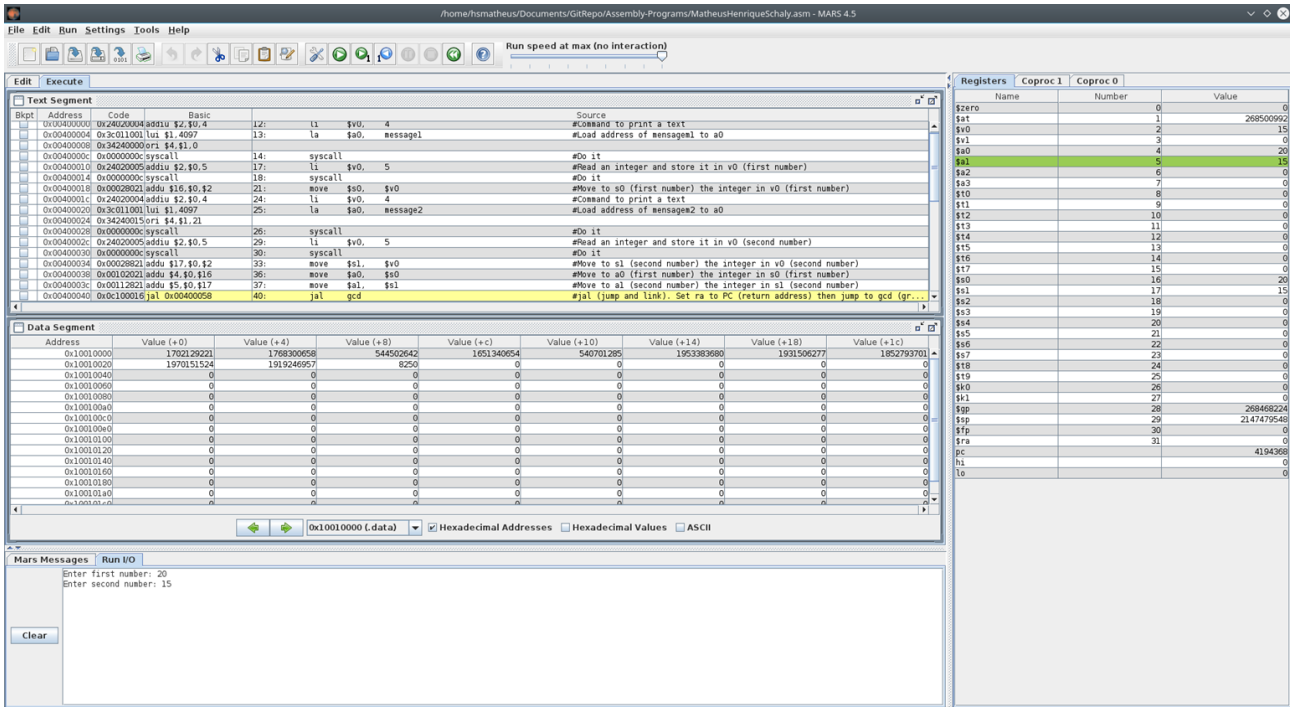
Início do programa. Foi requisitado o primeiro número inteiro ao usuário, e seu valor (20) foi armazenado no registrador $s0.



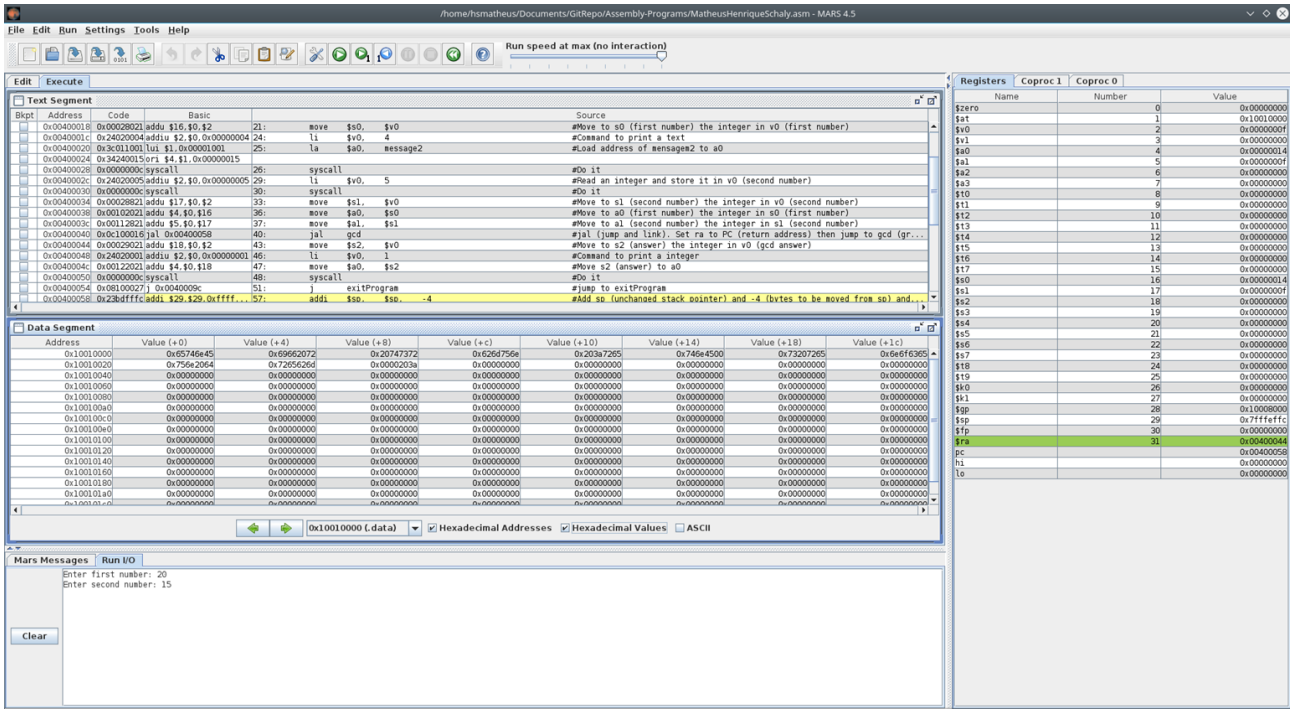
Em seguida, foi requisitado a entrada do segundo número inteiro, e seu valor (15) foi armazenado no registrador $s1.



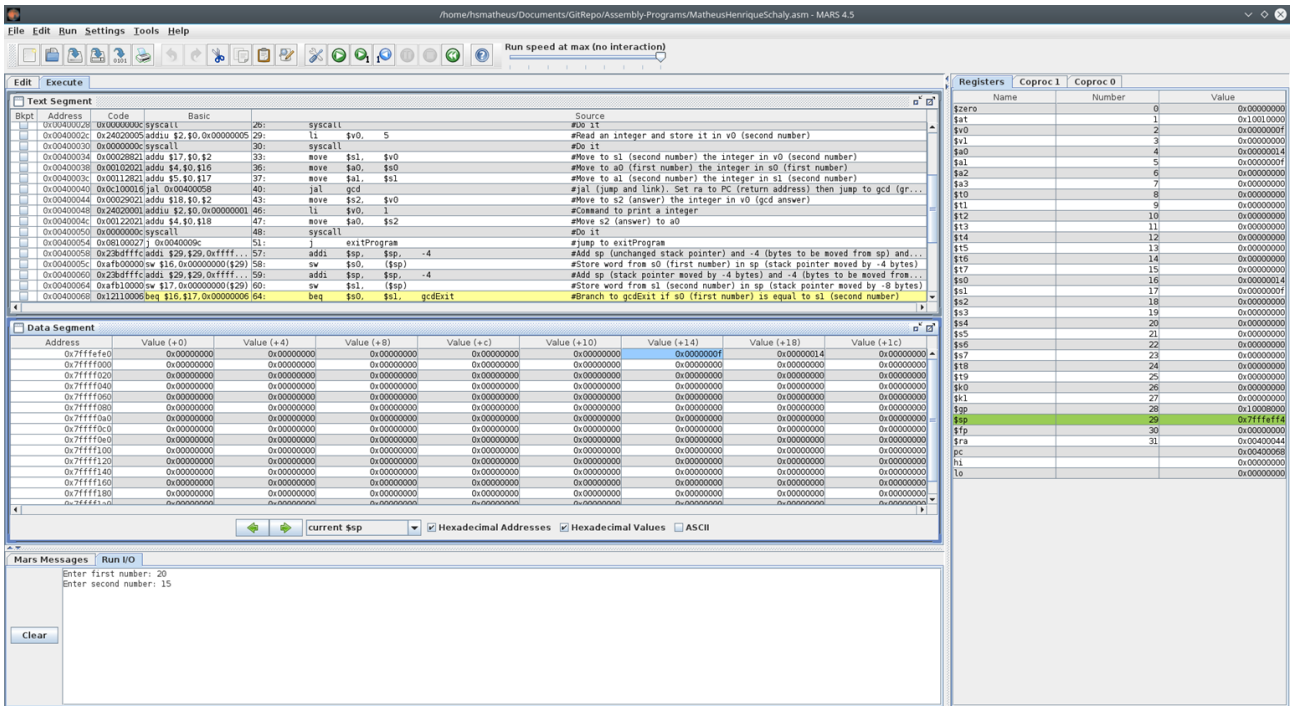
Os valores contidos em $s0 e $s1 foram transferidos respectivamente para os registradores $a0 e $a1, que são os registradores relativos aos argumentos do procedimento que será chamado.



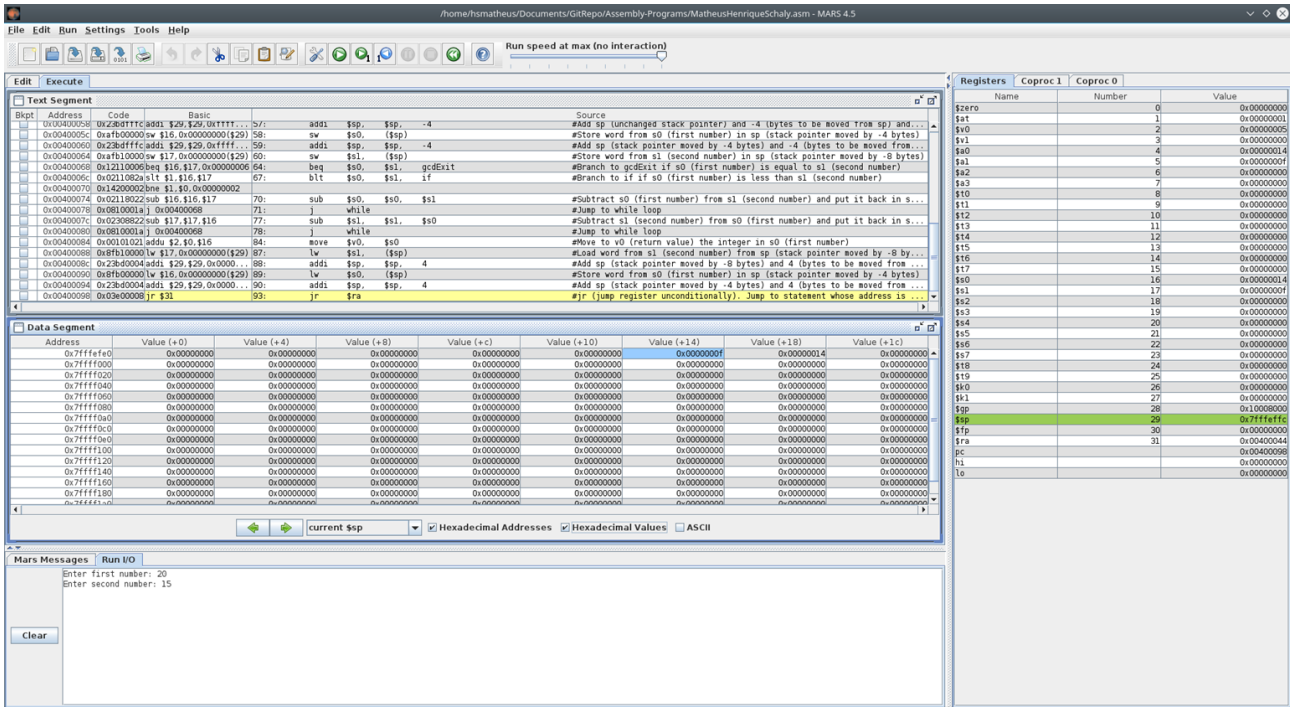
O procedimento foi invocado, usando o comando jal (jump and link), fazendo com que o registrador $ra recebesse o valor do PC de quando o comando jal foi chamado, somado de 4. Ou seja, o valor de PC no momento em que jal foi chamado era de 0x00400040, então $ra tornou-se 0x00400044.



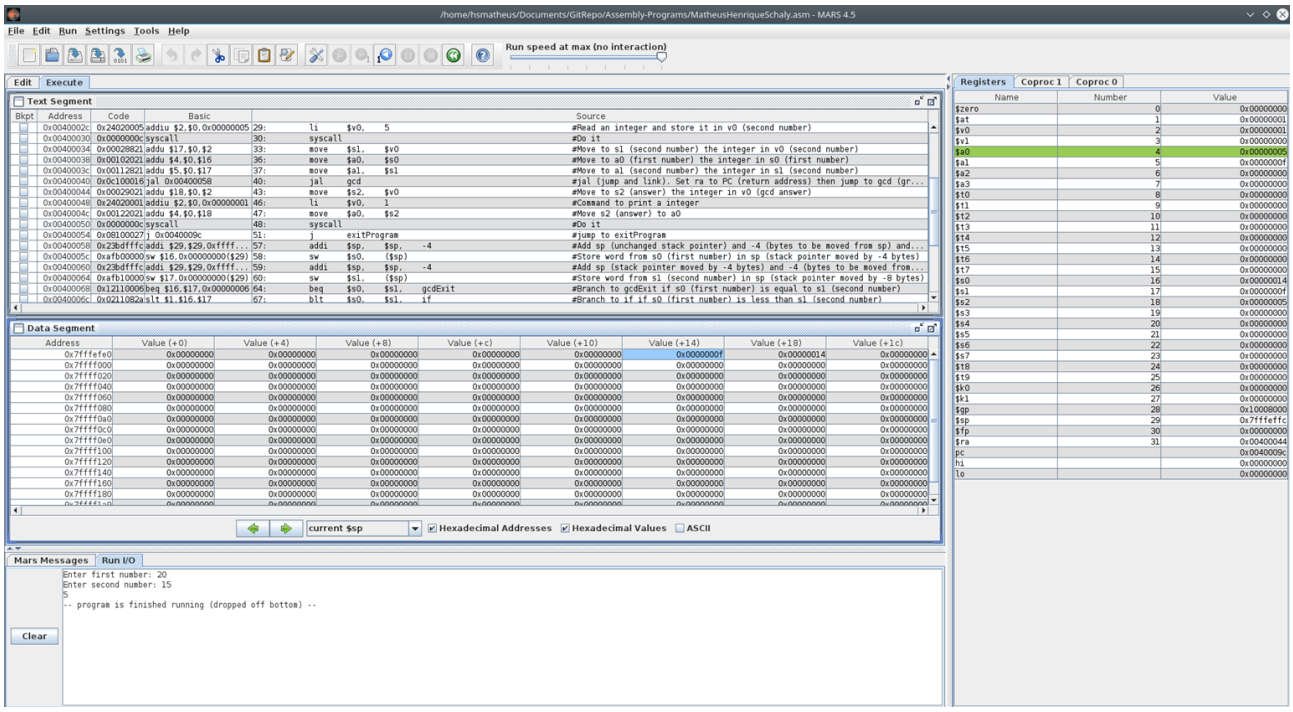
Os valores $s0 e $s1 foram empilhados no registrador $sp para que fossem mantidos inalterados ao final do procedimento. Para isso, o valor de $sp foi reduzido em 4 e recebeu $s0, em seguida, o valor $sp foi reduzido em 4 novamente e recebeu o valor $s1. Ou seja, os valores 15 e 20 foram armazenados no endereço 0x7fffefe0 + 14 e + 18 respectivamente.



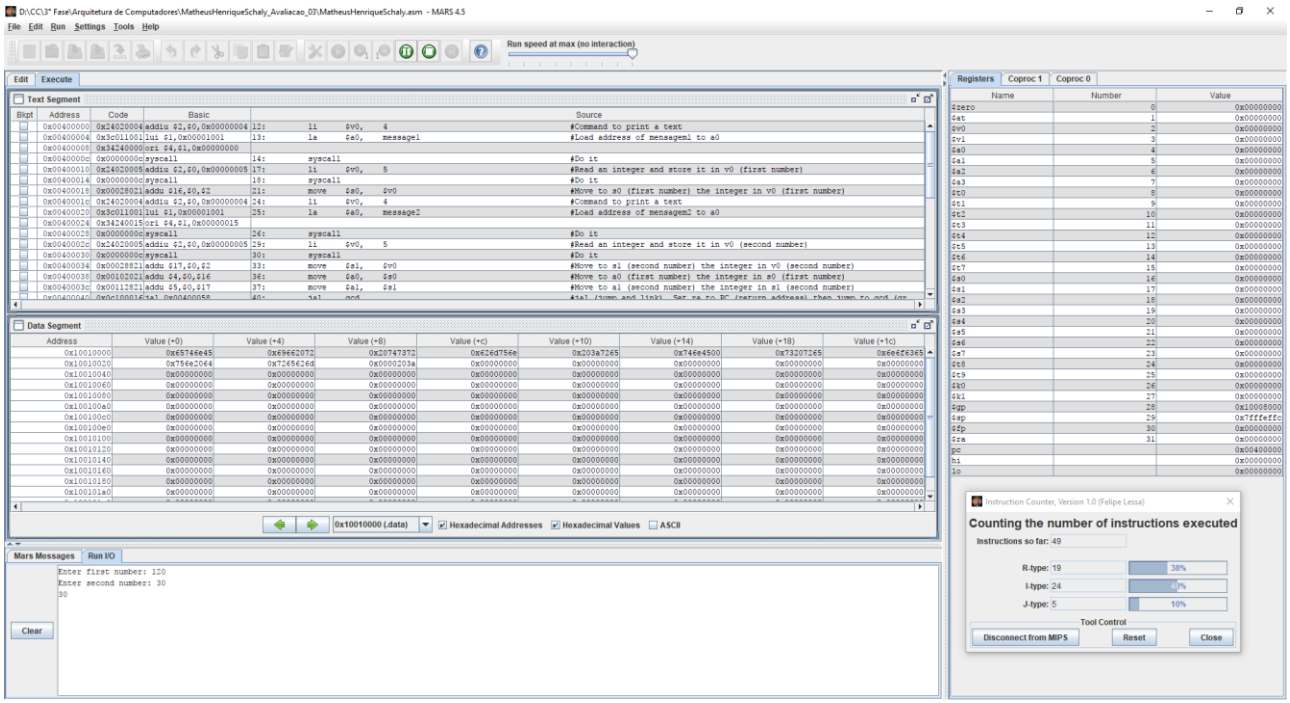
O loop foi realizado, o valor foi encontrado (5) e armazenado no registrador de retorno $v0. Em seguida, os valores originais de $s0 e $s1 foram desempilhados através do registrador $sp.



O programa retornou ao main através do comando jr (jump register), que retorna para o endereço referenciado por $ra (discutido anteriormente). Então, o resultado do programa (armazenado em $v0) foi transferido para $s2, que foi posteriormente impresso na saída do programa, finalizando-o.



Apresentação do contador de instruções, demonstrando a quantidade e porcentagem dos tipos de instruções realizadas do início ao fim da execução do programa.



Conclusão

Vimos a partir das capturas de tela, que o programa armazena corretamente os valores recebidos do usuário nos registradores $s0 e $s1, que foram transferidos para os registradores de argumento $a0 e $a1 respectivamente.

O procedimento é invocado, utilizando-se o comando jal, que manipulou o valor do registrador $ra (que será responsável pelo retorno do procedimento).

Em seguida, já dentro do procedimento, o empilhamento (utilizando-se o registrador $sp) dos valores $s0 e $s1 foi realizado, seguido pelo loop que calculou a resposta, desempilhamento dos valores e retorno para o main.

Finalmente, o resultado foi transferido de $v0 para $s2 que foi então apresentado na saída do programa, finalizando-o.