



UNIVALI

UNIVERSIDADE DO VALE DO ITAJAÍ
CAMPUS KOBRA SOL – 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.....	3
I – Código assembly MIPS.....	4
1 – Código.....	4
II – Resultados e análises.....	7
1 – Análise dos resultados.....	7
Conclusão.....	11

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.

The screenshot shows the MARS MIPS simulator interface. The Text Segment window displays the following assembly code:

```
0x00400000: 0x24020004 addiu $2,$0,4      12: li $v0, 4      #Command to print a text
0x00400004: 0x3c011001 lui $1,$1,0       13: la $a0, message1 #Load address of message1 to a0
0x00400008: 0x34240000 ori $4,$1,0        14: syscall        #Do it
0x0040000c: 0x00000000 syscall           15: syscall        #Read an integer and store it in v0 (first number)
0x00400010: 0x24020005 addiu $2,$0,5      16: li $v0, 5      #Do it
0x00400014: 0x00000000 syscall           17: syscall        #Read an integer and store it in v0 (second number)
0x00400018: 0x00028021 addu $16,$0,$2     18: move $s0, $v0  #Move to s0 (first number) the integer in v0 (first number)
0x0040001c: 0x24020004 addiu $2,$0,4      21: li $v0, 4      #Command to print a text
0x00400020: 0x3c011001 lui $1,$1,0       24: la $a0, message2 #Load address of message2 to a0
0x00400024: 0x34240000 ori $4,$1,0        25: syscall        #Do it
0x00400028: 0x00000000 syscall           26: syscall        #Read an integer and store it in v0 (second number)
0x0040002c: 0x24020005 addiu $2,$0,5      29: li $v0, 5      #Do it
0x00400030: 0x00000000 syscall           30: syscall        #Read an integer and store it in v0 (second number)
0x00400034: 0x00028021 addu $17,$0,$2     31: move $s1, $v0  #Move to s1 (second number) the integer in v0 (second number)
0x00400038: 0x00010021 addu $4,$0,$16     36: move $a0, $s0  #Move to a0 (first number) the integer in s0 (first number)
0x0040003c: 0x00011021 addu $5,$0,$17     37: move $a1, $s1  #Move to a1 (second number) the integer in s1 (second number)
0x00400040: 0x00012821 addu $5,$0,$17     37: move $a1, $s1  #Move to a1 (second number) the integer in s1 (second number)
0x00400044: 0x00010001 jal $ra            40: jal ocd        #jal fume and link. Set ra to PC (return address) then fume to ocd (ar...
```

The Registers window shows the initial values of the registers:

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	20
\$v1	3	0
\$a0	4	268500992
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	20
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$fp	29	2147479548
\$ra	30	0
\$pc	31	4194332

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

The screenshot shows the MARS MIPS simulator interface. The Text Segment window displays the same assembly code as the previous screenshot. The Registers window shows the updated values of the registers:

Name	Number	Value
\$zero	0	0
\$at	1	268500992
\$v0	2	15
\$v1	3	0
\$a0	4	268501013
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	20
\$s1	17	15
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$fp	29	2147479548
\$ra	30	0
\$pc	31	4194360

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Bkpt	Text Segment	Address	Code	Basic	n°
		0x00000000	0x40200000addiu \$2,\$0,4	12: li \$v0, 4 #Command to print a text	\$zero 0
		0x00400004	0x3c011001lui \$t1,4097	13: la \$a0, message1 #Load address of message1 to a0	\$t1 1
		0x00400008	0x34240000ori \$4,\$1,0		\$v0 2
		0x0040000c	0x00000000syscall		\$t1 3
		0x00400010	0x24020005addiu \$2,\$0,5	14: syscall #do it	\$a0 4
		0x00400014	0x00000000syscall	17: li \$v0, 5 #Read an integer and store it in v0 (first number)	\$a1 5
		0x00400018	0x00020002syscall	18: syscall #do it	\$a2 6
		0x0040001c	0x00020002addi \$16,\$0,\$2	21: move \$v0, \$0 #Move to \$0 (first number) the integer in v0 (first number)	\$a3 7
		0x00400020	0x3c011001lui \$t1,4097	24: li \$v0, 4 #Command to print a text	\$t0 8
		0x00400024	0x34240001ori \$4,\$1,21	25: la \$a0, message2 #Load address of message2 to a0	\$t1 9
		0x00400028	0x00000000syscall		\$t2 10
		0x0040002c	0x24020005addiu \$2,\$0,5	26: syscall #do it	\$t3 11
		0x00400030	0x00000000syscall	29: li \$v0, 5 #Read an integer and store it in v0 (second number)	\$t4 12
		0x00400034	0x00020002syscall	30: syscall #do it	\$t5 13
		0x00400038	0x00020002addi \$17,\$0,\$2	33: move \$s1, \$v0 #Move to s1 (second number) the integer in v0 (second number)	\$t6 14
		0x0040003c	0x00100021addiu \$4,\$0,\$16	36: move \$a0, \$s0 #Move to a0 (first number) the integer in v0 (first number)	\$t7 15
		0x00400040	0x00112021addi \$5,\$0,\$17	37: move \$a1, \$s1 #Move to a1 (second number) the integer in s1 (second number)	\$t8 16
		0x00400044	0x0c100016jalu \$g,0x00400058	40: jal gcd #jal (jump and link). Set ra to PC (return address) then jump to gcd (gr...	\$t9 17
					\$t0 18
					\$t1 19
					\$t2 20
					\$t3 21
					\$t4 22
					\$t5 23
					\$t6 24
					\$t7 25
					\$t8 26
					\$t9 27
					\$fp 28
					\$fp 29
					\$fp 30
					\$fp 31
					\$PC 4194368
					\$hi 0
					\$lo 0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1702129221	1768306958	544502642	1651340654	540701285	1953383680	1931506277	1852793701
0x10010004	1970151524	1919246957	0250	0	0	0	0	0
0x10010008	0	0	0	0	0	0	0	0
0x1001000c	0	0	0	0	0	0	0	0
0x10010010	0	0	0	0	0	0	0	0
0x10010014	0	0	0	0	0	0	0	0
0x10010018	0	0	0	0	0	0	0	0
0x1001001c	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010024	0	0	0	0	0	0	0	0
0x10010028	0	0	0	0	0	0	0	0
0x1001002c	0	0	0	0	0	0	0	0
0x10010030	0	0	0	0	0	0	0	0
0x10010034	0	0	0	0	0	0	0	0
0x10010038	0	0	0	0	0	0	0	0
0x1001003c	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010044	0	0	0	0	0	0	0	0

[illegible]

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 0x7ffefe0 + 14 e + 18 respectivamente.

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for the program. The Text Segment window shows the following instructions:

```
0x00400026: 0x00000000 syscall      #0 $t0
0x00400027: 0x24020005 addiu $t2, $0, 0x00000005
0x00400028: 0x00000000 syscall      #0 $t1
0x00400029: 0x00000000 syscall      #0 $t2
0x00400030: 0x00000000 syscall      #0 $t3
0x00400031: 0x00000000 syscall      #0 $t4
0x00400032: 0x00000000 syscall      #0 $t5
0x00400033: 0x00000000 syscall      #0 $t6
0x00400034: 0x00000000 syscall      #0 $t7
0x00400035: 0x00000000 syscall      #0 $t8
0x00400036: 0x00000000 syscall      #0 $t9
0x00400037: 0x00000000 syscall      #0 $t10
0x00400038: 0x00000000 syscall      #0 $t11
0x00400039: 0x00000000 syscall      #0 $t12
0x00400040: 0x00000000 syscall      #0 $t13
0x00400041: 0x00000000 syscall      #0 $t14
0x00400042: 0x00000000 syscall      #0 $t15
0x00400043: 0x00000000 syscall      #0 $t16
0x00400044: 0x00000000 syscall      #0 $t17
0x00400045: 0x00000000 syscall      #0 $t18
0x00400046: 0x00000000 syscall      #0 $t19
0x00400047: 0x00000000 syscall      #0 $t20
0x00400048: 0x00000000 syscall      #0 $t21
0x00400049: 0x00000000 syscall      #0 $t22
0x00400050: 0x00000000 syscall      #0 $t23
0x00400051: 0x00000000 syscall      #0 $t24
0x00400052: 0x00000000 syscall      #0 $t25
0x00400053: 0x00000000 syscall      #0 $t26
0x00400054: 0x00000000 syscall      #0 $t27
0x00400055: 0x00000000 syscall      #0 $t28
0x00400056: 0x00000000 syscall      #0 $t29
0x00400057: 0x00000000 syscall      #0 $t30
0x00400058: 0x00000000 syscall      #0 $t31
0x00400059: 0x00000000 syscall      #0 $t32
0x00400060: 0x00000000 syscall      #0 $t33
0x00400061: 0x00000000 syscall      #0 $t34
0x00400062: 0x00000000 syscall      #0 $t35
0x00400063: 0x00000000 syscall      #0 $t36
0x00400064: 0x00000000 syscall      #0 $t37
```

The Data Segment window shows the memory layout. The Registers window shows the state of registers \$zero through \$t0. The console shows the input 'Enter first number: 20' and 'Enter second number: 15'.

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.

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for the program. The Text Segment window shows the following instructions:

```
0x00400058: 0x28000000 addi $sp, $sp, -4
0x00400059: 0x00000000 syscall      #0 $t0
0x00400060: 0x28000000 addi $sp, $sp, -4
0x00400061: 0x00000000 syscall      #0 $t1
0x00400062: 0x28000000 addi $sp, $sp, -4
0x00400063: 0x00000000 syscall      #0 $t2
0x00400064: 0x28000000 addi $sp, $sp, -4
0x00400065: 0x00000000 syscall      #0 $t3
0x00400066: 0x28000000 addi $sp, $sp, -4
0x00400067: 0x00000000 syscall      #0 $t4
0x00400068: 0x28000000 addi $sp, $sp, -4
0x00400069: 0x00000000 syscall      #0 $t5
0x00400070: 0x28000000 addi $sp, $sp, -4
0x00400071: 0x00000000 syscall      #0 $t6
0x00400072: 0x28000000 addi $sp, $sp, -4
0x00400073: 0x00000000 syscall      #0 $t7
0x00400074: 0x28000000 addi $sp, $sp, -4
0x00400075: 0x00000000 syscall      #0 $t8
0x00400076: 0x28000000 addi $sp, $sp, -4
0x00400077: 0x00000000 syscall      #0 $t9
0x00400078: 0x28000000 addi $sp, $sp, -4
0x00400079: 0x00000000 syscall      #0 $t10
0x00400080: 0x28000000 addi $sp, $sp, -4
0x00400081: 0x00000000 syscall      #0 $t11
0x00400082: 0x28000000 addi $sp, $sp, -4
0x00400083: 0x00000000 syscall      #0 $t12
0x00400084: 0x28000000 addi $sp, $sp, -4
0x00400085: 0x00000000 syscall      #0 $t13
0x00400086: 0x28000000 addi $sp, $sp, -4
0x00400087: 0x00000000 syscall      #0 $t14
0x00400088: 0x28000000 addi $sp, $sp, -4
0x00400089: 0x00000000 syscall      #0 $t15
0x00400090: 0x28000000 addi $sp, $sp, -4
0x00400091: 0x00000000 syscall      #0 $t16
0x00400092: 0x28000000 addi $sp, $sp, -4
0x00400093: 0x00000000 syscall      #0 $t17
0x00400094: 0x28000000 addi $sp, $sp, -4
```

The Data Segment window shows the memory layout. The Registers window shows the state of registers \$zero through \$t0. The console shows the input 'Enter first number: 20' and 'Enter second number: 15'.

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.

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for a program. The Text Segment window shows the following code:

```

0x00400020: 0x24020005,addiu $2,$0,0x00000005 29: li $v0, 5 #Read an integer and store it in v0 (second number)
0x00400030: 0x0000000c,sycall 30: sycall #do it
0x00400034: 0x00020021,addu $17,$0,$2 31: move $s1, $v0 #Move to s1 (second number) the integer in v0 (second number)
0x00400038: 0x00010201,addu $4,$0,$16 36: move $a0, $s0 #Move to a0 (first number) the integer in s0 (first number)
0x0040003c: 0x00012821,addu $5,$0,$17 37: move $a1, $s1 #Move to a1 (second number) the integer in s1 (second number)
0x00400040: 0x00100016,jal 0x00400058 40: jal gcd #jal (jump and link): set ra to PC (return address) then jump to gcd (gr...)
0x00400044: 0x00029021,addu $18,$0,$2 43: move $s2, $v0 #Move to s2 (answer) the integer in v0 (gcd answer)
0x00400048: 0x24020001,addu $2,$0,0x00000001 46: li $v0, 1 #Command to print a integer
0x0040004c: 0x00120021,addu $4,$0,$18 47: move $a0, $s2 #Move s2 (answer) to a0
0x00400050: 0x0000000c,sycall 48: sycall #do it
0x00400054: 0x00100027,j 0x0040003c 51: j exitProgram #jump to exitProgram
0x00400058: 0x2800fffc,addi $29,$29,0xffff 57: addi $sp, $sp, -4 #Add sp (unchanged stack pointer) and -4 (bytes to be moved from sp) and...
0x0040005c: 0x00f00000,sw $16,0x00000000($29) 58: sw $s0, ($sp) #Store word from s0 (first number) in sp (stack pointer moved by -4 bytes)
0x00400060: 0x2800fffc,addi $29,$29,0xffff 59: addi $sp, $sp, -4 #Add sp (stack pointer moved by -4 bytes) and -4 (bytes to be moved from...
0x00400064: 0x00f00000,sw $17,0x00000000($29) 60: sw $s1, ($sp) #Store word from s1 (second number) in sp (stack pointer moved by -8 bytes)
0x00400068: 0x12110006,bq $16,$17,0x00000006 64: bq $s0, $s1, gcdExit #Branch to gcdExit if s0 (first number) is equal to s1 (second number)
0x0040006c: 0x0021082a,li $1,$16,$17 67: blt $s0, $s1, if #Branch to if if s0 (first number) is less than s1 (second number)
  
```

The Data Segment window shows memory addresses and values. The Registers window shows the state of registers. The Mars Messages window shows the input and output of the program.

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

The screenshot shows the Mars MIPS simulator interface. The main window displays the assembly code for a program. The Text Segment window shows the following code:

```

0x00400000: 0x24020005,addiu $2,$0,0x00000005 12: li $v0, 4 #Command to print a test
0x00400004: 0x3c011001,li $1,0x00000101 13: la $a0, message1 #Load address of message1 to a0
0x00400008: 0x0040000c,sycall 14: sycall #do it
0x0040000c: 0x0000000c,sycall 14: sycall #do it
0x00400010: 0x24020005,addiu $2,$0,0x00000005 15: li $v0, 5 #Read an integer and store it in v0 (first number)
0x00400014: 0x0000000c,sycall 18: sycall #do it
0x00400018: 0x00020021,addu $16,$0,$2 21: move $a0, $v0 #Move to a0 (first number) the integer in v0 (first number)
0x0040001c: 0x24020004,addiu $2,$0,0x00000004 24: li $v0, 4 #Command to print a test
0x00400020: 0x3c011001,li $1,0x00000101 25: la $a0, message2 #Load address of message2 to a0
0x00400024: 0x0040000c,sycall 26: sycall #do it
0x00400028: 0x24020005,addiu $2,$0,0x00000005 29: li $v0, 5 #Read an integer and store it in v0 (second number)
0x00400030: 0x0000000c,sycall 30: sycall #do it
0x00400034: 0x00020021,addu $17,$0,$2 33: move $a1, $v0 #Move to a1 (second number) the integer in v0 (second number)
0x00400038: 0x00020021,addu $4,$0,$16 36: move $a0, $s0 #Move to a0 (first number) the integer in s0 (first number)
0x0040003c: 0x00012821,addu $5,$0,$17 37: move $a1, $s1 #Move to a1 (second number) the integer in s1 (second number)
0x00400040: 0x00100016,jal 0x00400058 40: jal gcd #jal (jump and link): set ra to PC (return address) then jump to gcd (gr...)
  
```

The Data Segment window shows memory addresses and values. The Registers window shows the state of registers. The Mars Messages window shows the input and output of the program. The Instruction Counter window is open, showing the number of instructions executed for each type:

Instruction Type	Count	Percentage
R-type	19	38%
I-type	24	47%
J-type	5	10%

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.