Universidade do Vale do Itajaí
Campus Kobrasol

Arquitetura de Computadores

Avaliação 03

Grupo: Mauricio Macário de Farias Junior

Professor : Douglas Rossi de Melo

São José, 2017, 06 de outubro

Resumo

Neste trabalho será desenvolvido um programa que descobre o máximo divisor comum a partir de um procedimento que executa um loop fazendo a diminuição do maior pelo menor sucessivamente até que os dois sejam iguais retornando assim a resposta e a imprimindo para o usuário.

O programa utiliza a linguagem de Assembly do MIPS.

Desenvolvimento

Aqui será apresentado o código e capturas de telas, mostrando seu funcionamento:

Código em Assembly MIPS

```
.data

message1: .asciiz "Primeiro numero: "
message2: .asciiz   "Segundo numero: "
message3: .asciiz   "Maximo divisor comum: "
error_message: .asciiz "Numero invalido!!\n"
```

Declaração das constantes usadas para imprimir mensagens.

```
main:
          j    numero1
```

Esse jump é necessário por causa da posição do rotulo error1.

```
error1:
          li   $v0, 4
          la   $a0, error_message
          syscall
```

Impressão da mensagem de erro.

```
numero1:
          li   $v0, 4
          la   $a0, message1
          syscall

          li   $v0, 5
          syscall
          blt  $v0, 1, error1
          add  $s0, $zero, $v0
          j    numero2
```

Loop que lê e verifica se o número passado é valido, caso não seja valido volta para o rótulo error1, senão armazena o valor no registrador $s0 e vai para a próxima parte do código.

```
error2:
          li   $v0, 4
          la   $a0, error_message
          syscall
numero2:
          li   $v0, 4
          la   $a0, message2
          syscall

          li   $v0, 5
          syscall
          blt  $v0, 1, error2
          add  $s1, $zero, $v0
```

Mesmo loop que o numero1, apenas muda para qual registrador ele armazena o valor.

```
          add  $a0, $zero, $s0
          add  $a1, $zero, $s1
          jal  proc_mdc
```

Chamada do procedimento proc_mdc, passando os argumentos para os registradores de argumento ($a0, $a1), armazenando o endereço da próxima instrução em $ra, e fazendo o jump ao rotulo do procedimento.

```
proc_mdc:
          addi $sp, $sp, -4
          sw   $s0, 0($sp)
          addi $sp, $sp, -4
          sw   $s1, 0($sp)
          add  $s0, $zero, $a0
          add  $s0, $zero, $a1
```

Empilhamento dos registradores que serão usados dentro do método, guardando seus valores em $sp, e jogando os valores dos argumentos em $s0, e $s1.

```
while:
          beq  $s0, $s1, end_while
          bgt  $s0, $s1, if_greater
```

Checagem dos números para saber se são iguais ou não, se forem iguais apenas sai do programa e retorna o valor do registrador.

```
if_less:
          sub  $s1, $s1, $s0
          j    while
```
Caso o primeiro número passado ($s0) seja menor que o segundo ($s1), diminui o primeiro do segundo.

```
if_greater:
          sub  $s0, $s0, $s1
          j    while
```
Caso o primeiro número passado ($s0) seja maior que o segundo ($s1), diminui o segundo do primeiro.

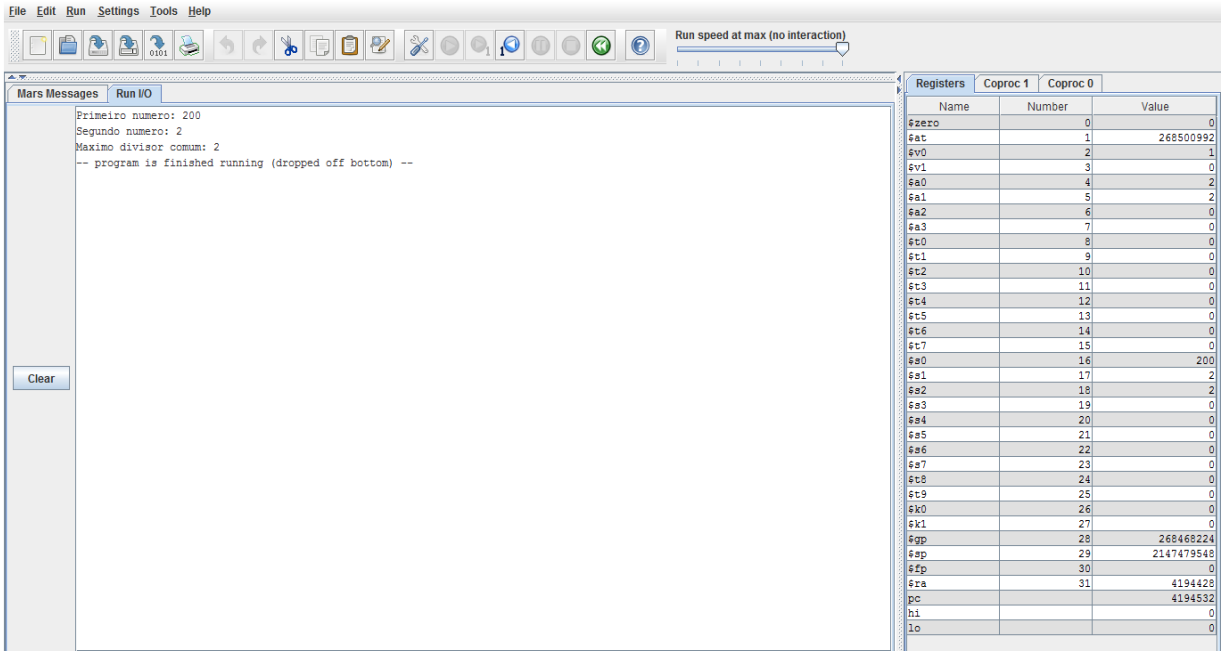
```
end_while:
          add  $v0, $zero, $s0
          lw   $s1, 0($sp)
          addi $sp, $sp, 4
          lw   $s0, 0($sp)
          addi $sp, $sp, 4
          jr   $ra
```
Término do procedimento e do loop, jogando o resultado final para o registrador de retorno ($v0), desempilhando os registradores empilhados no começo do procedimento e voltando ao endereço armazenado em $ra.

```
          add  $s2, $zero, $v0

          li   $v0, 4
          la   $a0, message3
          syscall

          li   $v0, 1
          add  $a0, $zero, $s2
          syscall

          j    exit
```
Copia valor retornado para o registrador $s2, imprime o valor, e sai do programa.

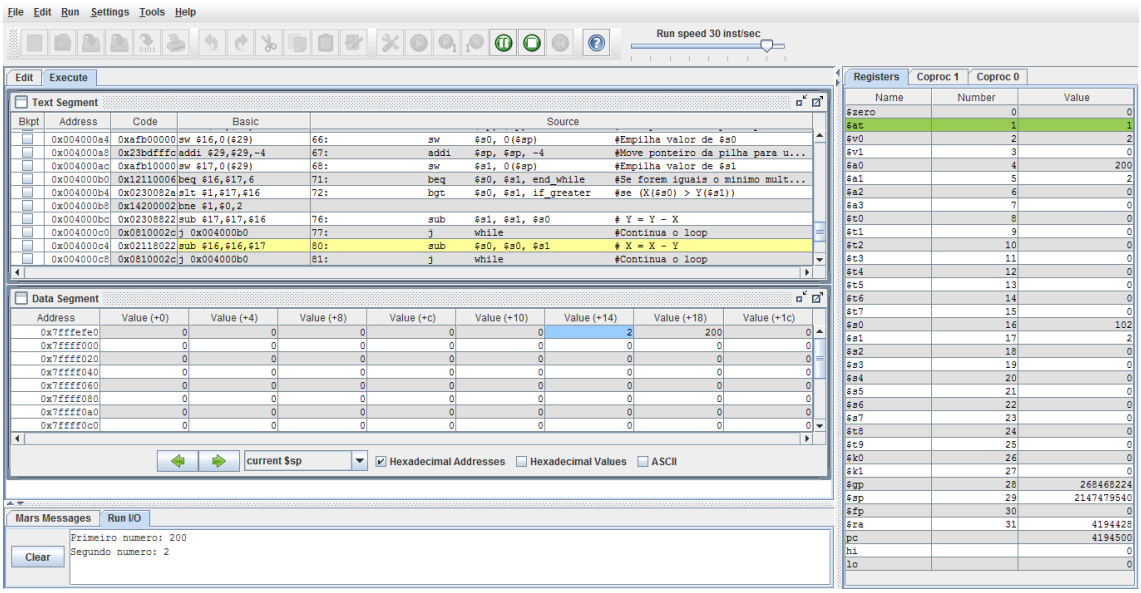
Código em C++

```
int proc_mdc(int x, int y) {
   while (x != y) {
      if (x < y) {
         y = y - x;
      else
         x = x - y;
   }
   return x;
}
```

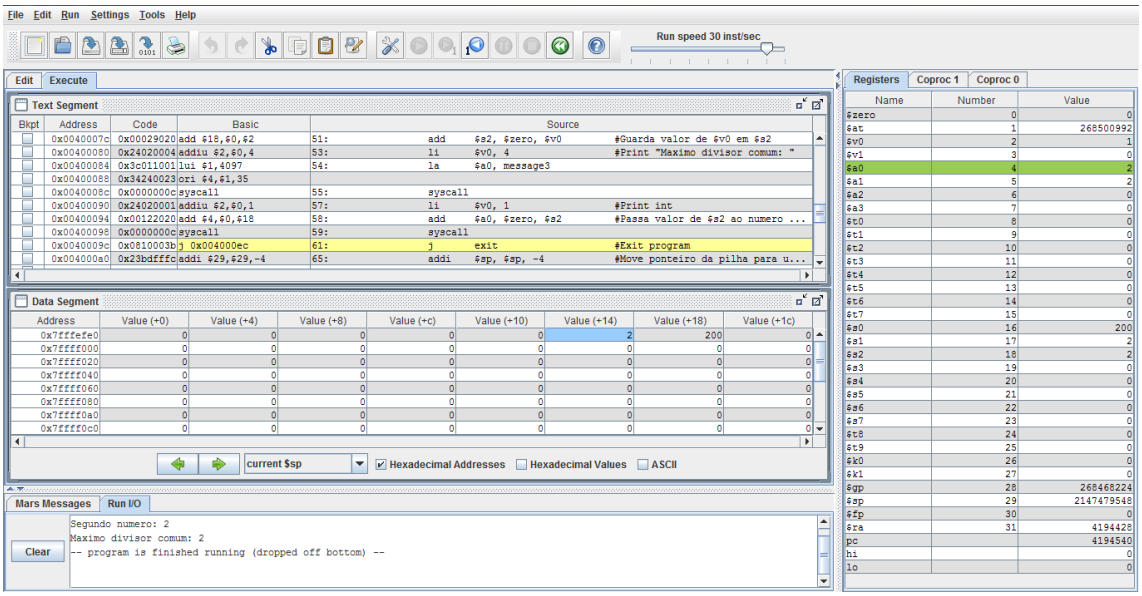
Resultados



Acima está apresentado as entradas e as saidas do programa.

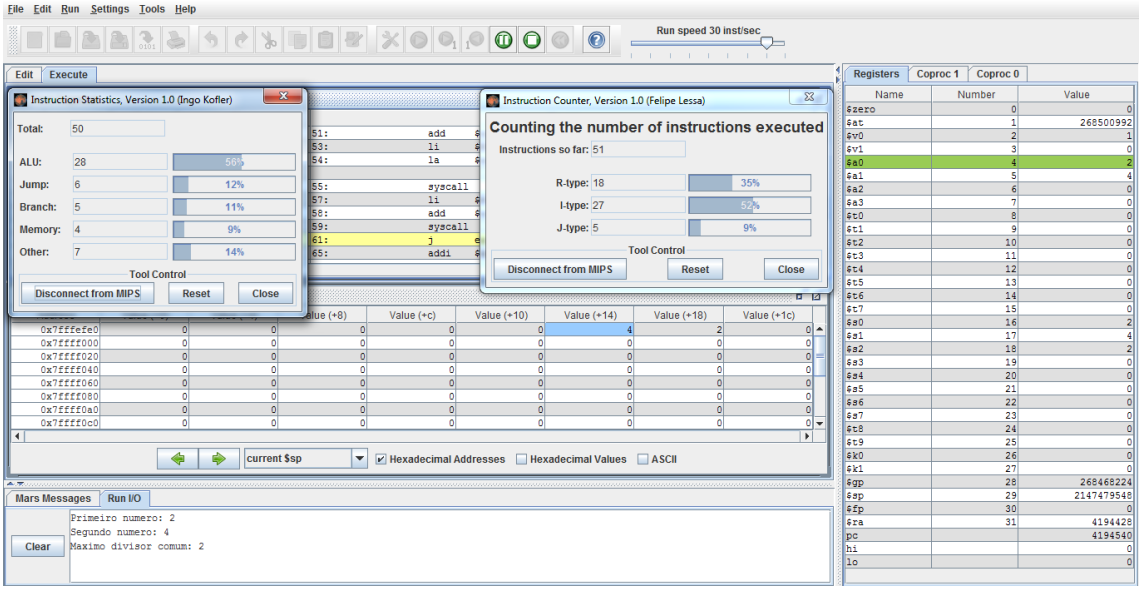


Acima está apresentado uma captura de tela durante o processo de subtração do programa, perceba que $a0 e $a1 não são modificados durante o processo, isso se deve ao fato de usar-se os registradores $s0 e $s1 dentro do loop, empilhando-os no começo do procedimento para que ao final do mesmo eles voltem aos seus valores originais.



Acima está apresentado o estado final do programa, quando sua execução termina, perceba que o registrador $s0 volta ao seu valor original, isso deve-se ao desempilhamento ao final do procedimento retornando os valores a seus estados originais, nota-se também que o valor final do resultado encontra-se no registrador $s2, isso foi necessário pois o registrador $v0 foi utilizado para as chamadas de sistema executadas pelo syscall necessárias para imprimir o valor no console.

Quadro de instruções



Acima esta apresentado o quadro de instruções, mostrando à esquerda o quadro apresentado pela ferramenta *Instruction Statistics* do MIPS e à direita o quadro apresentado pela ferramenta *Instruction Counter* do MIPS.

Conclusão

Foi observado que o programa funciona de forma corretamente, empilhando os registrador $s0 e $s1, para assim poder passar os valores passados nos registradores de argumento ($a0 e $a1) para os registradores $s0 e $s1 respectivamente, no programa em questão não se vê muita utilidade para essa movimentação de valores, mas foi usada para simular uma situação em que o procedimento não sabe em quais registradores estão localizados os valores, em teoria o procedimento teria ciência apenas dos valores passados aos registradores de argumento.

O empilhamento funciona corretamente, não modificando so valores dos registradores empilhados ao término do programa.

Ao final do programa foi retornado o valor de retorno ao respectivo registrador($v0) e o processo do programa principal continuou normalmente imprimindo o valor calculado e retornado a partir do procedimento.