SPIM

O SPIM é um simulador que executa programas escritos para os processadores MIPS R2000 e R3000. Ele é capaz de ler e executar arquivos em linguagem de montagem do MIPS. Um manual detalhado deste simulador feito pelo próprio autor pode ser encontrado aqui.

A seguir serão apresentadas rapidamente algumas de suas principais funções e será demonstrada a sua utilização em exemplos práticos, para que este possa ser usado na elaboração do Projeto 2.

Para que você possa se familiarizar com o funcionamento do software, veja antes familiarizando-se com a interface do SPIM (apenas o PCSPIM).

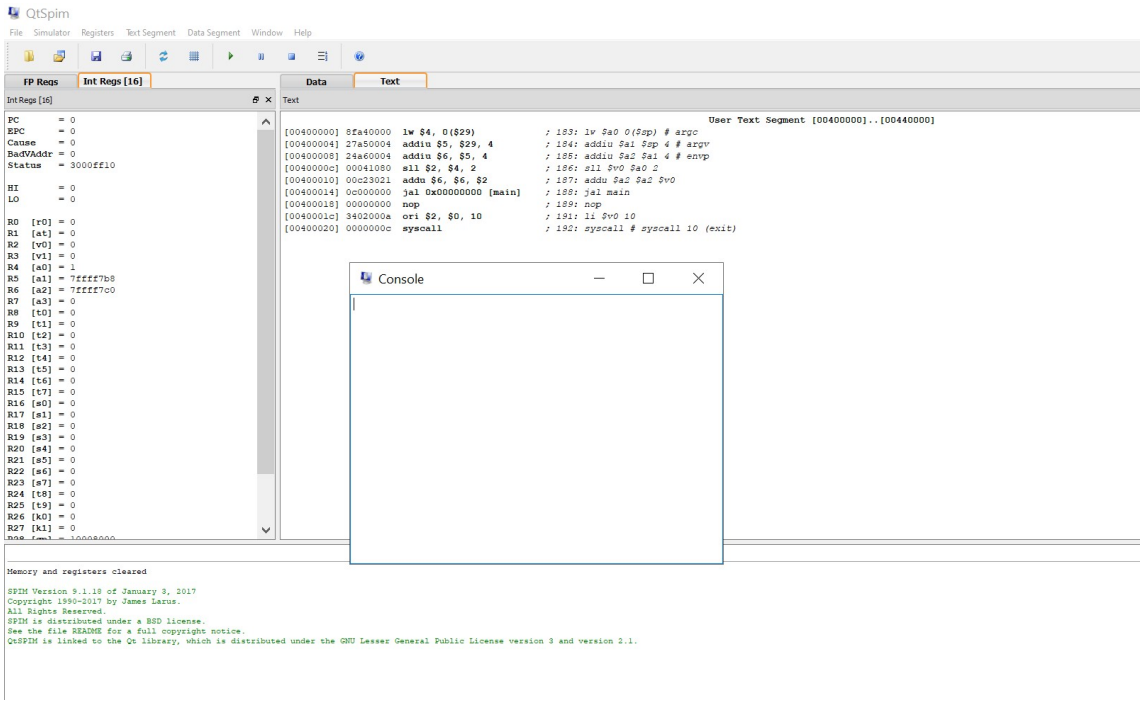
Para executar seus programas escritos em linguagem de montagem do MIPS, estes devem ser feitos em um editor de texto comum, como o notepad.exe. Estes arquivos deverão ser salvos com a extensão "*.s*" ou "*.asm*" para depois serem abertos e executados pelo SPIM.

A seguir são apresentados alguns exemplos de programas que podem auxiliar no desenvolvimento do projeto:

Exemplo 1 - Trabalhando com registradores

Exemplo 2 - Trabalhando com dados em memória

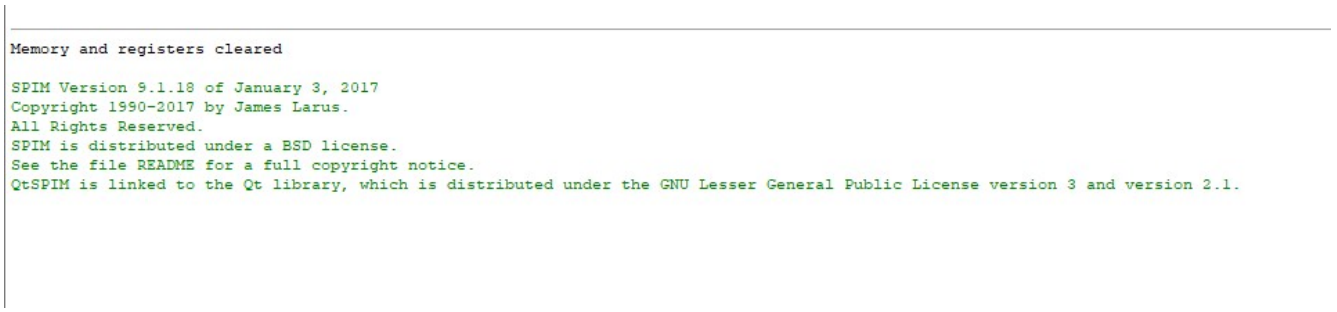
Familiarizando-se com o SPIM



O QtSpim é composto por cinco janelas principais, que são:

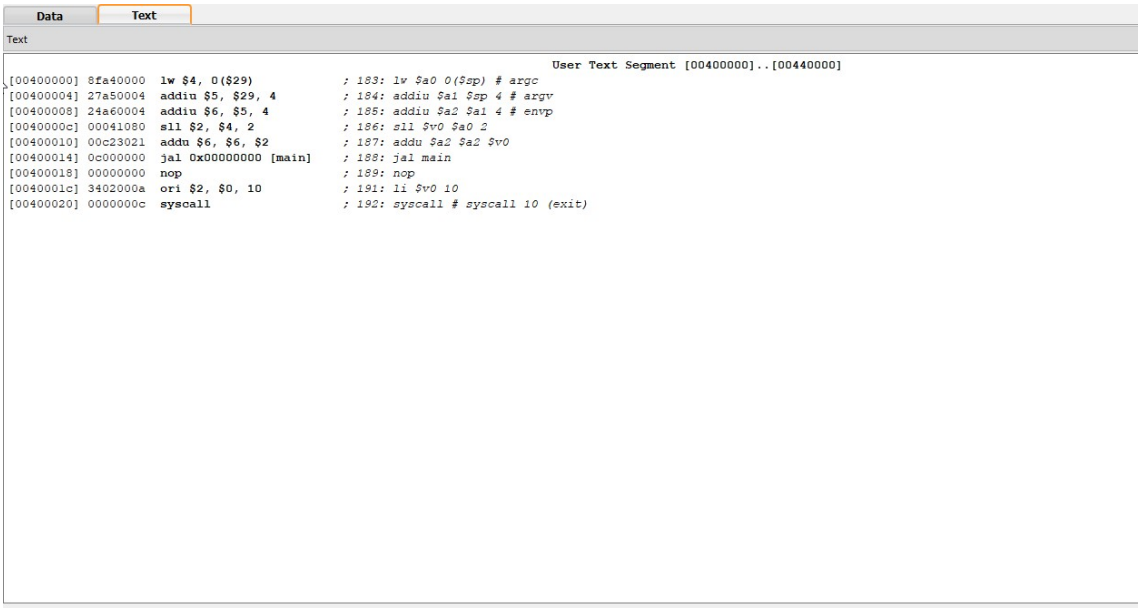
1. Messages

Contém as mensagens geradas pelo Spim para o usuário. Geralmente são apresentadas mensagens sobre o carregamento do programa ou a execução do mesmo e erros ocorridos se for o caso.



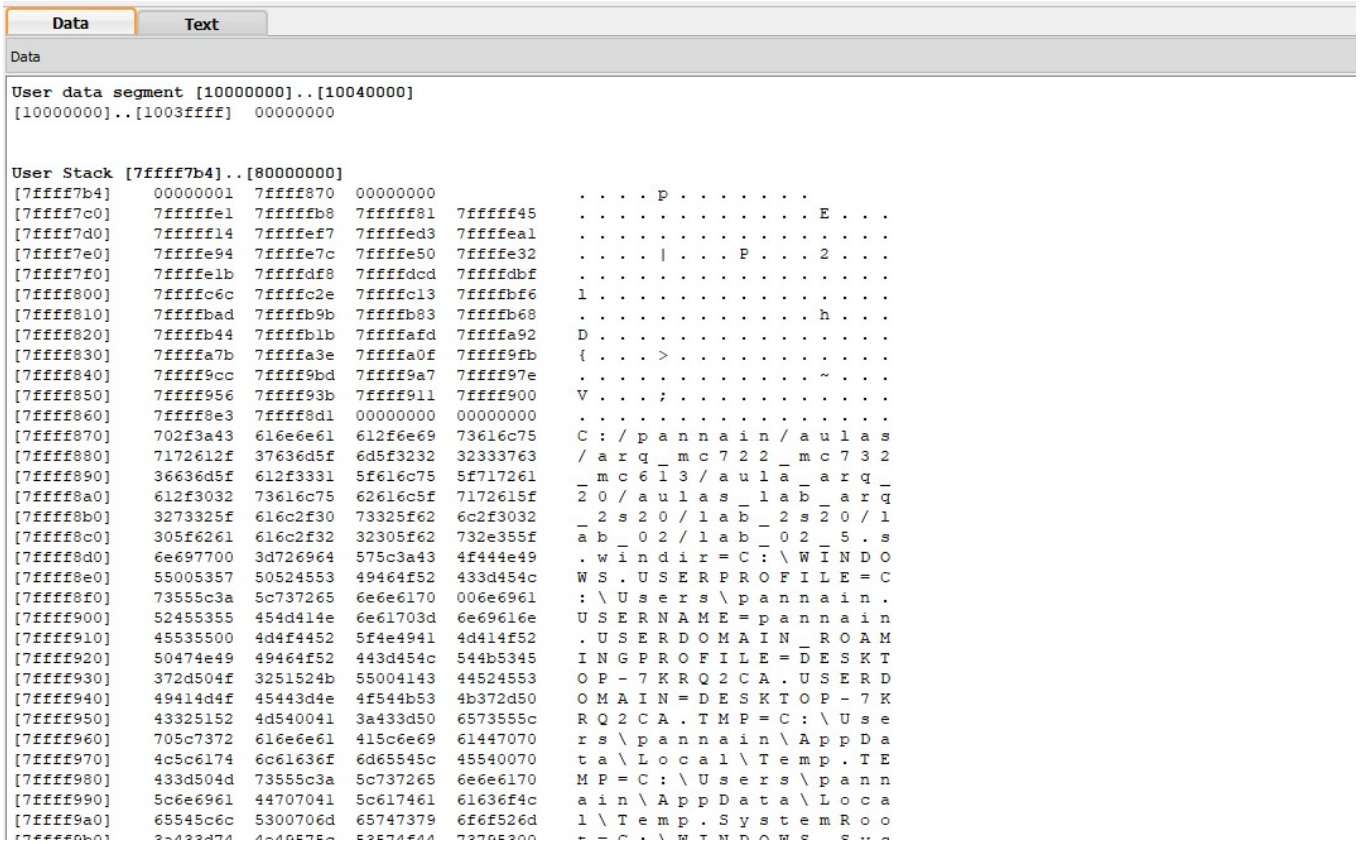
2. Text Segment

Nesta janela é mostrado as instruções que foram carregadas em memória. As instruções aparecem em duas colunas, a da direita para o código que foi carregado e a da esquerda para as instruções geradas pelo Spim.



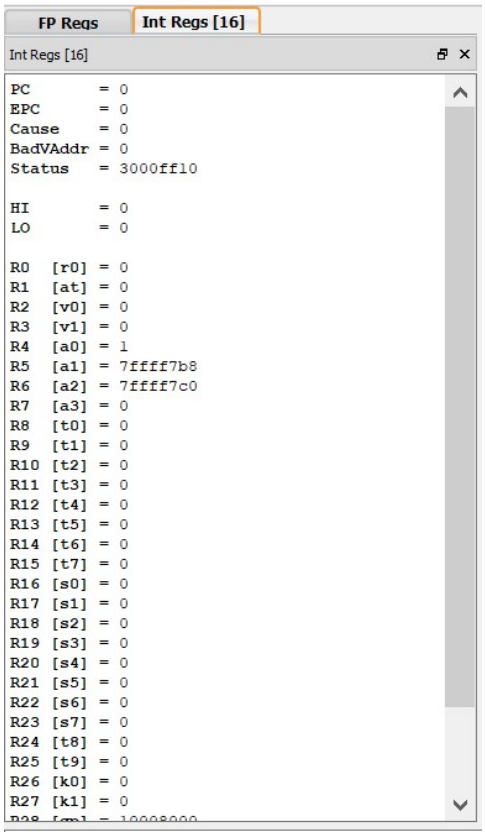
3. Data Segment

Mostra os dados carregados em memória e os dados da pilha.



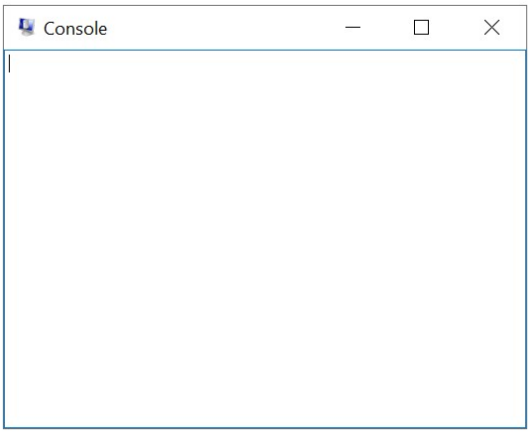
4. Registers

Esta janela mostra os valores armazenados em todos os registradores do MIPS, incluindo os da unidade de ponto flutuante (FPU).



5. Console

No Spim é possível usar uma espécie de "console" para exibir mensagens e receber entrada de dados.



Serviços do Sistema

O Spim possui uma lista de serviços implementados para auxiliar principalmente na interface com o usuário por meio de um console. Para utilizar um desses serviços, basta colocar em $v0 o código do serviço, definir os parâmetros (se houverem) e em seguida utilizar a instrução syscall.

Serviço	**Código**	**Parâmetros**	**Resultados**	**Descrição**
print_int	1	$a0 = integer		Escreve um valor inteiro no console.
print_float	2	$f12 = float		
print_double	3	$f12 = double		
print_string	4	$a0 = string		Escreve uma string no console.
read_int	5		integer (em $v0)	Lê um valor inteiro entrado no console.
read_float	6		float (em $f0)	
read_double	7		double (em $f0	
read_string	8	$a0 = buffer, $a1 = length		Lê uma string do console (n caracteres)
sbrk	9	$a0 = quantidade	endereço (em $v0)	Retorna um ponteiro para um bloco de memória contendo n bytes)
exit	10			Encerra a execução do programa.

Exemplo 1 - Trabalhando com registradores

Abra o notepad.exe (ou outro editor se preferir). Executaremos o exemplo apresentado a seguir, cujo objetivo é realizar a operação *f = (g + h) - (i + j)*.

Detalhe importante: Para ques os programas possam ser executados no SPIM, deverá **sempre** ser definido onde o programa deverá ser iniciado. Isto é feito através da declaração do símbolo *main,* da seguinte maneira:

```
.text                    # indica que as linhas seguintes contém instruções
.globl main              # define o símbolo main como sendo global
main:                    # indica o início do programa
```

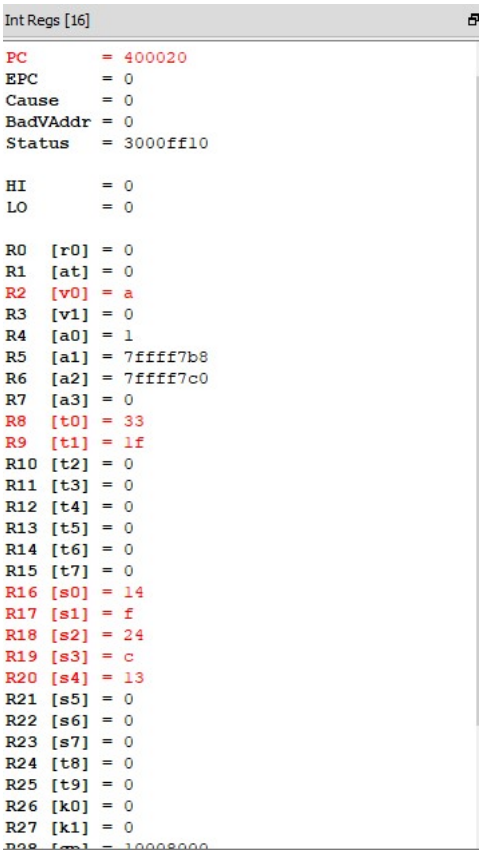
Agora podemos entrar com as instruções do programa. De início, precisamos atribuir valores para as variáveis *g, h, i* e *j* (registradores $s1, $s2, $s3 e $s4 respectivamente). Para isso entraremos com o seguinte código:

```
li $s1,15                # registrador $s1 contém o valor imediato 15
li $s2,36                # registrador $s2 contém o valor imediato 36
addi $s3, $zero, 12      # registrador $s3 contém o valor imediato 12
addi $s4, $zero, 19      # registrador $s4 contém o valor imediato 19
```

O uso das instruções *li* e *addi* foi proposital para demonstrar essas duas formas de carga de valores imediatos em registradores. Agora vamos executar a operação colocando o resultado em $s0.

```
add $t0,$s1,$s2          # registrador $t0 contém g + h
add $t1,$s3,$s4          # registrador $t1 contém i + j
sub $s0,$t0,$t1          # registrador $s0 contém (g + h) - (i + j)
```

Salve o arquivo com o nome de "*arq1.s*" e abra-o no SPIM através do menu File, Open. Abra a janela dos Registradores no menu Window, Registers. Observe que os registradores estão todos zerados. Agora clique no menu Simulator, e simule. Isto irá executar o código. Observe agora a mudança no estado dos registradores (em vermelho).



O $s0 contém o resultado da nossa operação: 14h (os valores dos registradores estão todos em hexadecimal). Conferindo: (15+36) = 51; (12+19) = 31; (51-31) = 20 = (14h).

Exemplo 2 - Trabalhando com dados em memória

Detalhe importante: O acesso à memória do SPIM em nossos programas deve ser feito com valores acima da posição inicial do global pointer (10008000h). Esta é a parte da memória do SPIM que iremos utilizar para armazenar nossos dados. Mais a frente veremos também como armazenar dados "constantes" de uma forma mais prática, sem a necessidade de utilizar as instruções de store.

Vamos fazer um exercício simples de acesso a memória:

Tendo-se um array de 100 elementos (words) que inicia no endereço de memória 5000 (em direção aos endereços crescentes) transfira este array para o endereço 6000.

Primeiro precisamos carregar na memória este array de elementos que foi considerado no enunciado. Como as posições de memória 5000 e 6000 estão fora da área que temos acesso na memória, utilizaremos o $gp (global pointer) + 5000 como endereço inicial do array fonte. Já para o array destino, utilizaremos o valor de $gp + 6000.

```
.text
.globl main
main:
move $s0,$gp
addi $s0,$s0,5000          # Ponteiro para os dados do array fonte ($gp) + 5000
move $s2,$s0
addi $s2,$s2,400           # Marcador para indicar o final do array $gp + (100posições de 4bytes)
```

Desta forma temos os ponteiros necessários para trabalhar com o primeiro array. O $s0 será o ponteiro e será incrementado sempre em 4 posições para apontar para a próxima word (próximo elemento). Vamos armazenar nele um dado qualquer, como por exemplo um valor incrementado sempre em 9 (9, 18, 27, 36...)

Detalhe importante: Para marcar pontos importantes no programa, utilizamos "labels" (rótulos). É através deles que executamos funções como jump e branch. Para definir um label, coloque um identificador seguido do sinal de dois pontos, ex: "repetir:" e escreva o código. Neste exercício faremos um loop para preencher os 100 elementos do array, por isso vamos precisar de um label para chamar a cada iteração. Quando o label é chamado (através de um bne por exemplo) a próxima instrução a ser executada é a da linha seguinte ao label.

```
li $t0,9                   # Carrega no reg. temporário $t0 um valor para ser armazenado no array
dados:
sw $t0,0($s0)              # Armazena o valor na posição do
```

```
                                array apontada por $s0
addi $s0,$s0,4                  # Aponta para a próxima posição no
                                array (incrementa em 4 o ponteiro)
addi $t0,$t0,9                  # Altera o valor a ser armazenado
                                no array (incrementa em 9)
bne $s0,$s2,dados               # Enquanto não chegar ao fim do
                                array, repete o laço
```

Vamos executar esta primeira parte do programa para testarmos o armazenamento dos valores do array. Salve o arquivo com o nome de "*arq_2a.s*" e abra-o no SPIM. Execute o código (F5 e depois OK). Vamos verificar se ocorreu tudo bem. Na janela dos registradores o $s0 deverá estar em 10009518h que é a marca do final do array (o array inicia em 10009388h e o último elemento está em 10009514Ch). O registrador $t0 contém o valor 38Dh (909 em decimal), ou seja, o valor que seria armazenado na posição seguinte a última. Até aqui tudo Ok.

Agora verificaremos os valores em memória. Abra a janela Data Segment (Window, Data Segment). Ela deverá estar assim:

Data

```
User data segment [10000000]..[10040000]
[10000000]..[100081f3]  00000000
[100081f4]    00000009  00000012  0000001b
[10008200]    00000024  0000002d  00000036  0000003f
[10008210]    00000048  00000051  0000005a  00000063
[10008220]    0000006c  00000075  0000007e  00000087
[10008230]    00000090  00000099  000000a2  000000ab
[10008240]    000000b4  000000bd  000000c6  000000cf
[10008250]    000000d8  000000e1  000000ea  000000f3
[10008260]    000000fc  00000105  0000010e  00000117
[10008270]    00000120  00000129  00000132  0000013b
[10008280]    00000144  0000014d  00000156  0000015f
[10008290]    00000168  00000171  0000017a  00000183
[100082a0]    0000018c  00000195  0000019e  000001a7
[100082b0]    000001b0  000001b9  000001c2  000001cb
[100082c0]    000001d4  000001dd  000001e6  000001ef
[100082d0]    000001f8  00000201  0000020a  00000213
[100082e0]    0000021c  00000225  0000022e  00000237
[100082f0]    00000240  00000249  00000252  0000025b
[10008300]    00000264  0000026d  00000276  0000027f
[10008310]    00000288  00000291  0000029a  000002a3
[10008320]    000002ac  000002b5  000002be  000002c7
[10008330]    000002d0  000002d9  000002e2  000002eb
[10008340]    000002f4  000002fd  00000306  0000030f
[10008350]    00000318  00000321  0000032a  00000333
[10008360]    0000033c  00000345  0000034e  00000357
[10008370]    00000360  00000369  00000372  0000037b
[10008380]    00000384  00000000  00000000  00000000
[10008390]..[1003ffff]  00000000
```

Aqui o SPIM apresenta os dados carregados em memória. Perceba que ele mostra somente as posições ocupadas. Os dados no array devem ser os seguintes: o valor 9 na primeira posição, o valor 18 na segunda e assim sucessivamente até o valor 900 na última posição. Podemos conferir os valores armazenados no nosso array: 9h, 12h, 1Bh... = 9, 18, 27... O último valor é 384h = 900.

Agora podemos continuar com o exercício. Vamos fazer a cópia dos dados para o array destino.

```
move $s0,$gp
addi $s0,$s0,5000               # Definimos novamente o ponteiro para os
                                dados do array fonte ($gp + 5000)
move $s1,$gp
addi $s1,$s1,6000               # Ponteiro para os dados do array destino
```

```
                                ($gp + 6000)
transfere:
lw $t0,0($s0)                   # Armazena em t0 o conteúdo da posição
                                apontada por $s0 (array fonte)
sw $t0,0($s1)                   # Armazena no array destino (apontado por
                                $s1) o valor carregado
addi $s0,$s0,4                  # Incrementa s0 em 4 (para chegar-se ao
                                próximo elemento no array fonte)
addi $s1,$s1,4                  # Incrementa s1 em 4 (para chegar-se ao
                                próximo elemento no array destino)
bne $s0,$s2,transfere           # Enquanto s0 não chegar em 400 (100
                                elementos), repete o laço
```

Salve novamente o arquivo ("*arq*_2b.s") e execute-o.

Agora a janela de dados vai apresentar os dois arrays, sendo que o segundo foi armazenado da posição 10009770h para cima. Observe que esta posição é o $gp (10008000h) + 6000.

```
User data segment [10000000]..[10040000]
[10000000]..[10009387]  00000000
[10009388]    00000009  00000012
[10009390]    0000001b  00000024  0000002d  00000036
[100093a0]    0000003f  00000048  00000051  0000005a
[100093b0]    00000063  0000006c  00000075  0000007e
[100093c0]    00000087  00000090  00000099  000000a2
[100093d0]    000000ab  000000b4  000000bd  000000c6
[100093e0]    000000cf  000000d8  000000e1  000000ea
[100093f0]    000000f3  000000fc  00000105  0000010e
[10009400]    00000117  00000120  00000129  00000132
[10009410]    0000013b  00000144  0000014d  00000156
[10009420]    0000015f  00000168  00000171  0000017a
[10009430]    00000183  0000018c  00000195  0000019e
[10009440]    000001a7  000001b0  000001b9  000001c2
[10009450]    000001cb  000001d4  000001dd  000001e6
[10009460]    000001ef  000001f8  00000201  0000020a
[10009470]    00000213  0000021c  00000225  0000022e
[10009480]    00000237  00000240  00000249  00000252
[10009490]    0000025b  00000264  0000026d  00000276
[100094a0]    0000027f  00000288  00000291  0000029a
[100094b0]    000002a3  000002ac  000002b5  000002be
[100094c0]    000002c7  000002d0  000002d9  000002e2
[100094d0]    000002eb  000002f4  000002fd  00000306
[100094e0]    0000030f  00000318  00000321  0000032a
[100094f0]    00000333  0000033c  00000345  0000034e
[10009500]    00000357  00000360  00000369  00000372
[10009510]    0000037b  00000384  00000000  00000000
```

Pronto! Os dados foram transferidos para o array destino.