



**Aluno:** Bruno Cordeiro Mendes

**Matrícula:** 15/0007094

## Trabalho II: Simulador MIPS

### Descrição do Problema:

Como foi visto na disciplina MIPS, acrônimo para Microprocessor without interlocked pipeline stages (microprocessador sem estágios intertravados de pipeline), é uma arquitetura de microprocessadores RISC desenvolvida pela MIPS Computer Systems. E para se criar um código executável em um MIPS é necessário o uso de um software montador denominado MARS no qual a linguagem utilizada no MARS para ser criado esse código executável é o Assembly.

O problema proposto no trabalho em questão foi a criação de um programa em uma linguagem de alto nível, do qual foi escolhida a **linguagem C**, que simulasse um processador MIPS apartir de dois arquivos (data e text) com os segmentos de instrução e os segmentos de dados necessários gerados pelo MARS.

Cada instrução do MIPS é composta por 32 bits, dos quais são divididos entre 2 a 6 segmentos que especificam quais manipulações devem ser feitas na memória e no banco de processadores para que a instrução seja realizada.

Para se construir o simulador foi necessária a especificação das instruções do MIPS utilizando um recurso da linguagem C denominada “enum”:

```
32 enum OPCODES { // so sao considerados os 6 primeiros bits dessas constantes
33
34     EXT=0x00, LH=0x21, SB=0x28, BLEZ=0x06, SLTIU=0x0B,
35     LHU=0x25, SH=0x29, BGTZ=0x07, ANDI=0x0C, JAL=0x03,
36     BEQ=0x04, ADDI=0x08, ORI=0x0D, J=0x02, LW=0x23,
37     LBU=0x24, SW=0x2B, LB=0x20, LUI=0x0F, ADDIU=0x09,
38     BNE=0x05, SLTI=0x0A, XORI=0x0E
39 };
40
41 enum FUNCT{
42     ADD=0x20, OR=0x25, SLL=0x00, SUB=0x22, XOR=0x26, SRL=0x02,
43     MULT=0x18, NOR=0x27, SRA=0x03, DIV=0x1A, AND=0x24, SLT=0x2A,
44     JR=0x08, SYSCALL=0x0c, MFHI=0x10, MFL0=0x12, ADDU=0x21
45 };
46
```

### Funções implementadas:

As funções implementadas são compostas de 2 conjuntos, sendo elas as funções já implementadas no trabalho 1 e as novas funções para a execução correta da simulação.

- **void fetch();**  
A função void fetch() lê uma instrução da memória e coloca-a em ri, atualizando o pc para apontar para a próxima instrução (soma 4).
- **void decode();**  
Deve extrair todos os campos da instrução:
  - opcode: código da operação
  - rs: índice do primeiro registrador fonte
  - rt: índice do segundo registrador fonte

- rd: índice do registrador destino, que recebe o resultado da operação
- shamnt: quantidade de deslocamento em instruções shift e rotate
- funct: código auxiliar para determinar a instrução a ser executada
- k16: constante de 16 bits, valor imediato em instruções tipo I
- k26: constante de 26 bits, para instruções tipo J

- **void execute():**

A função void execute() executa a instrução que foi lida pela função fetch() e decodificada por decode(). Dentro dessa função foi utilizado um switch para o opcode e o funct, com o intuito de realizar a instrução decodificada.

- **void step():**

A função step() executa uma instrução do MIPS: step() => fetch(), decode(), execute().

- **void run():**

A função run() executa o programa até encontrar uma chamada de sistema para encerramento, ou até o pc ultrapassar o limite do segmento de código (2k words).

- **void dump\_mem(int inicio, int fim, char format);**

Imprime o conteúdo da memória a partir do endereço start até o endereço end. O formato pode ser em hexa ('h') ou decimal ('d').

- **void dump\_reg(char format);**

Imprime o conteúdo dos registradores do MIPS, incluindo o banco de registradores e os registradores pc, hi e lo. O formato pode ser em hexa ('h') ou decimal ('d').

- **int32\_t lb(uint32\_t address, int16\_t kte);**

Função *load byte*.

- **int32\_t lh(uint32\_t address, int16\_t kte);**

Função *load half word*.

- **int32\_t lw(uint32\_t address, int16\_t kte);**

Função *load word*.

- **int32\_t lbu(uint32\_t address, int16\_t kte);**

Função *load byte unsigned*.

- **int32\_t lhu(uint32\_t address, int16\_t kte);**

Função *load half word unsigned*.

- **void sb(uint32\_t address, int16\_t kte, int8\_t dado);**

Função *store byte*.

- **void sh(uint32\_t address, int16\_t kte, int16\_t dado);**

Função *store half word*.

- **void sw(uint32\_t address, int16\_t kte, int32\_t dado);**

Função *store word*.

- **void menu():**

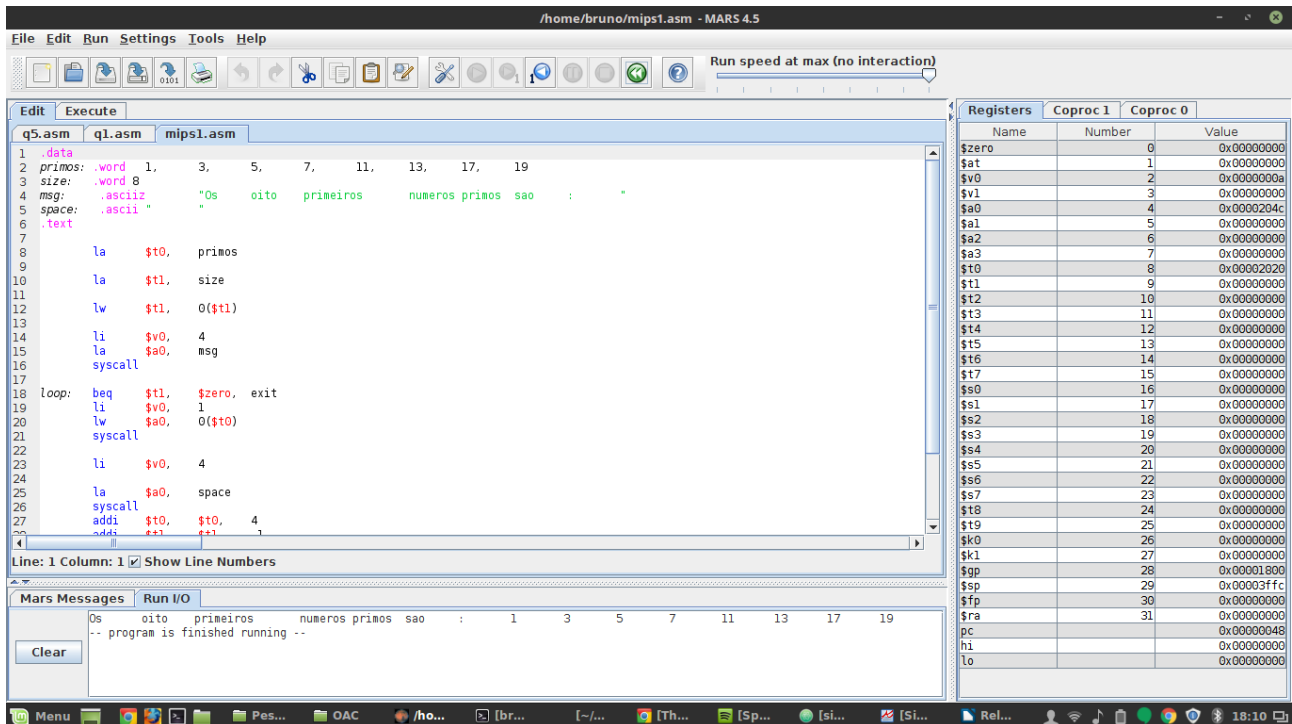
Executa menu iterativo para que o usuário possa realizar o dump da memória ou do banco de registradores após a execução do programa.

## Testes e Resultados:

Para testar o simulador foi utilizado o código “asm” encontrado na especificação do trabalho e e outros dois códigos feitos utilizando funções recursivas.

### - Oito primeiros números primos.

A função deste programa é simplesmente imprimir um vetor contendo os 8 primeiros números primos:



```
1 .data
2 primos: .word 1, 3, 5, 7, 11, 13, 17, 19
3 size: .word 8
4 msg: .asciiz "Os oito primeiros numeros primos sao : "
5 space: .asciiz " "
6 .text
7
8 la $t0, primos
9
10 la $t1, size
11
12 lw $t1, 0($t1)
13
14 li $v0, 4
15 la $a0, msg
16 syscall
17
18 loop: beq $t1, $zero, exit
19 li $v0, 1
20 lw $a0, 0($t0)
21 syscall
22
23 li $v0, 4
24 la $a0, space
25 syscall
26 addi $t0, $t0, 4
27 addi $t1, $t1, 1
```

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x0000204c
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00002020
\$t1	9	0x00000008
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x00001800
\$sp	29	0x00003ffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00000048
hi		0x00000000
lo		0x00000000

Mars Messages

```
Os oito primeiros numeros primos sao : 1 3 5 7 11 13 17 19
-- program is finished running --
```

E como resultado da simulação obteve-se:

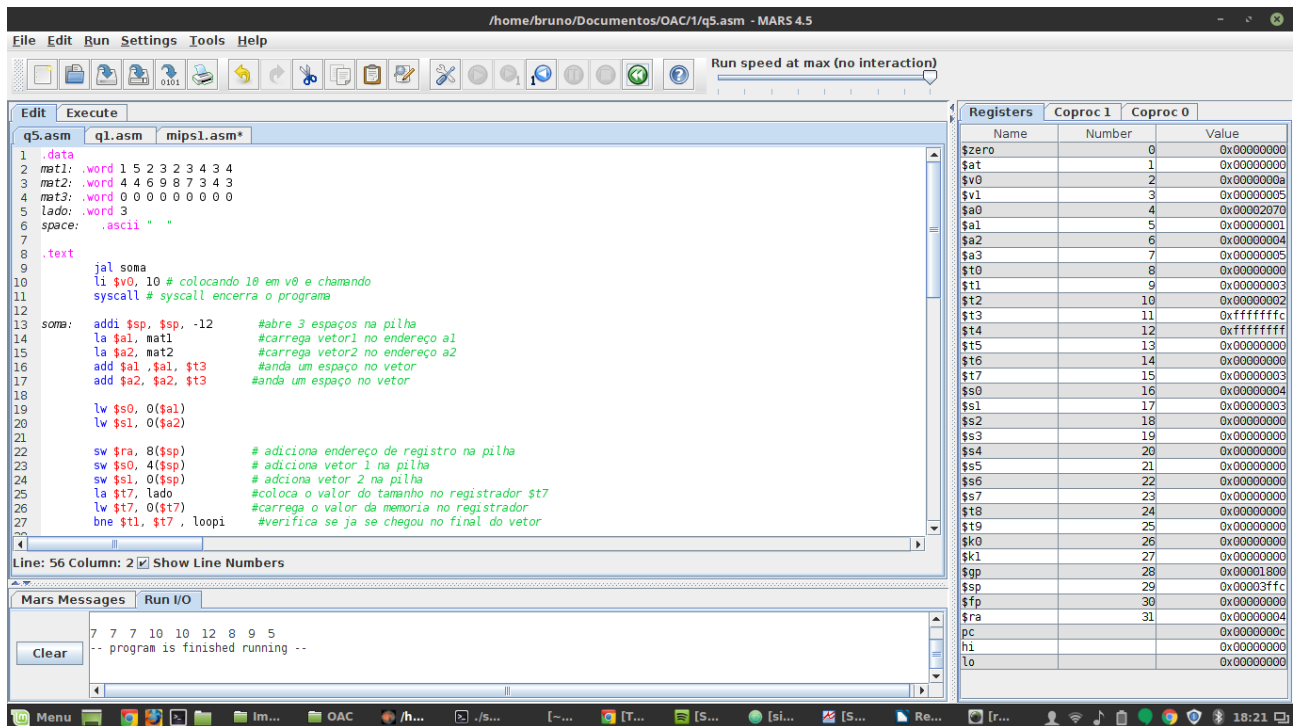


```
Arquivo Editar Ver Pesquisar Terminal Ajuda
+ OAC gcc simulador.c -o simulador
+ OAC ./simulador
Os oito primeiros numeros primos sao : 1 3 5 7 11 13 17 19
-- program is finished running --

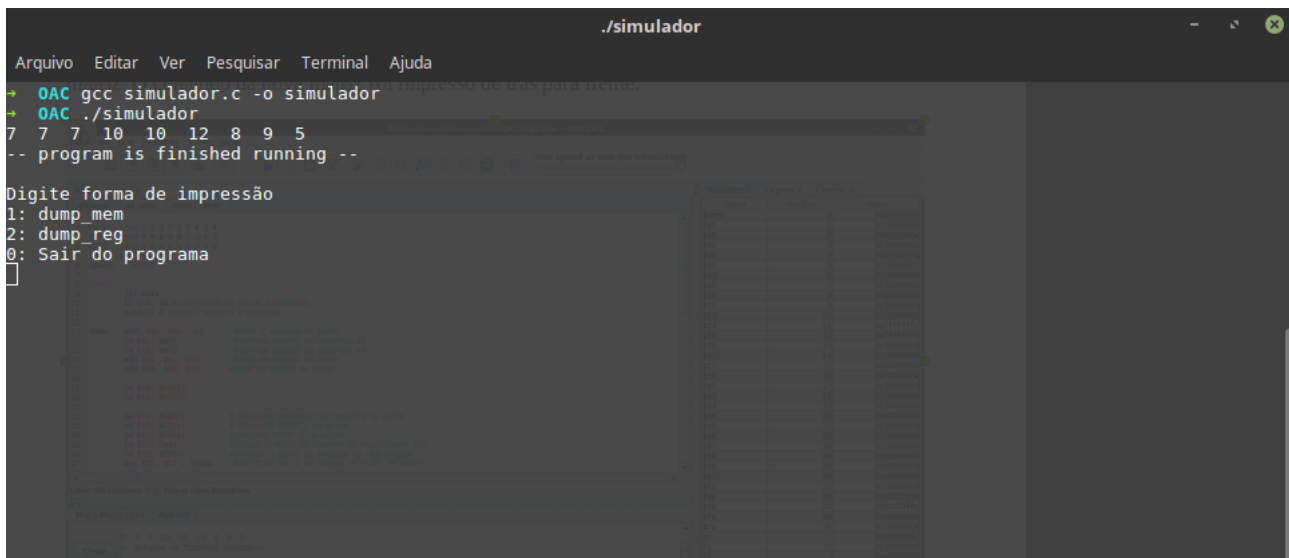
Digite forma de impressão
1: dump_mem
2: dump_reg
0: Sair do programa
█
```

## - Soma de matrizes

Neste programa foi realizada a soma de duas matrizes e seu armazenamento em uma terceira matriz. O conteúdo da nova matriz foi impresso de trás para frente:

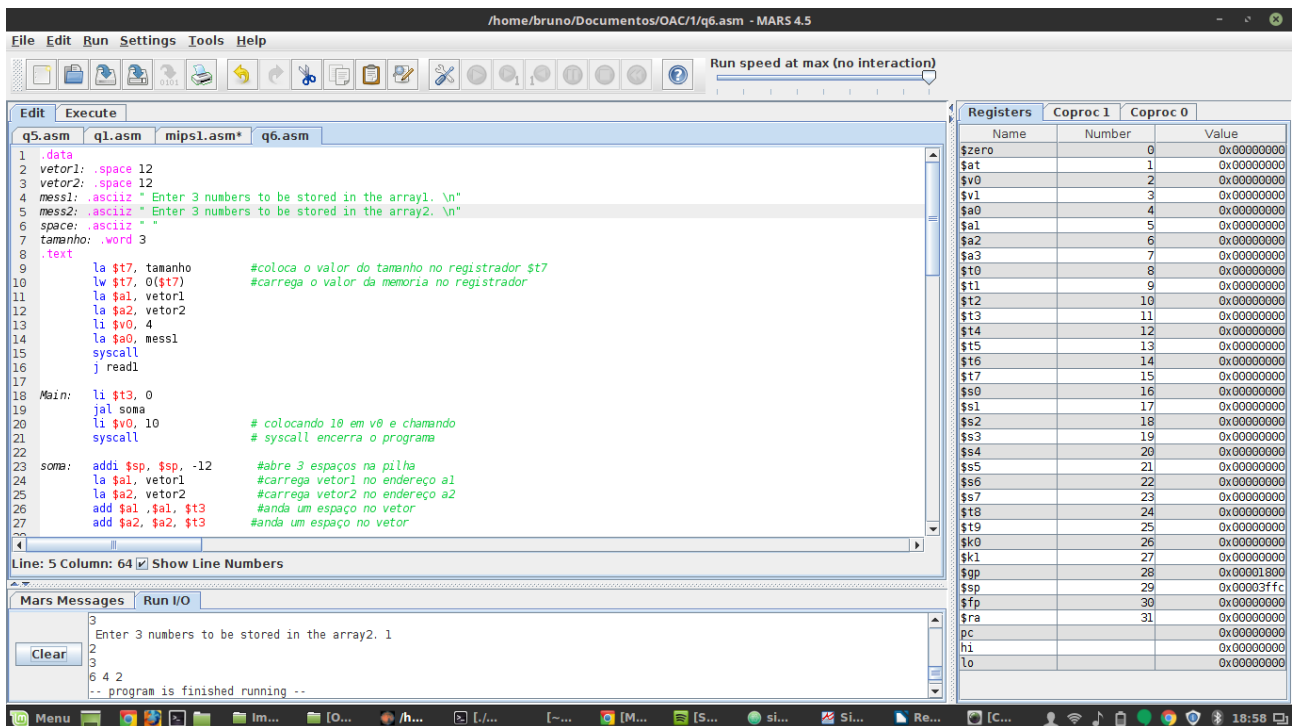


Como resultado da simulação obteve-se:

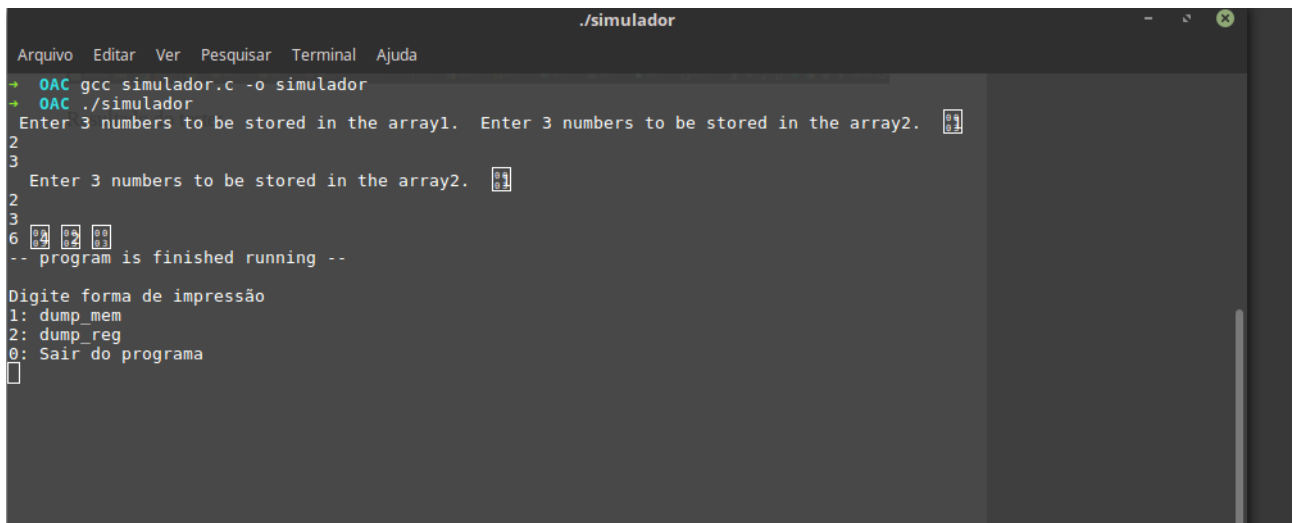


## -Soma de vetores, valores dados pelo usuário.

Nesse programa em questão o usuário deve inserir 3 valores para cada array, o resultado também é impresso de trás para frente:



Resultado do teste:



## Informações adicionais:

- Linguagem utilizada no trabalho: C.
- Ambiente: Linux 64x 8GB de RAM.
- Editor de texto utilizado: Sublime Text 3.1.1.
- Para executar o arquivo digite no terminal:  
gcc simulador.c -o simulador  
./simulador