Universidade de Brasília 01/2015

Organização e Arquitetura de Computadores

Aluna: Marina Martins de Miranda

Matrícula: 11/0132351

Trabalho 2

A plataforma utilizada foi: CodeBlocks gcc 4.7.1 Windows/unicode - 32 bit .

Objetivo

Este trabalho constitui a implementação de um simulador da arquitetura MIPS na linguagem C. Há a implementação das funções de busca da instrução (fetch()), decodificação da instrução (decode()) e execução da instrução (execute()). O programa binário executado é gerado a partir do montador MARS, por meio do "Dump Memory". O simulador lê os arquivos binários contendo o segmento de código e o segmento de dados para sua memória e o executa.

Para tanto, foi criado um vetor para representar a memória

```
#define MEM_SIZE 4096

int32_t mem[MEM_SIZE];
```

O tamanho 4096 é em palavras, porque cada elemento do vetor é um int32_t, com 32 bits ou 4 bytes. Assim, a memória é de 4096 palavras ou 4096*4 = 16384 bytes, sendo metade destinada à área de código, mem[0] a mem [2047], e metade destinada à área de dados, mem[2048] a mem[4095].

Também foi criado um vetor para representar o banco de registradores

```
int32_t breg[BREG_SIZE];
```

E também os registradores especiais foram criados: pc, ri, hi,lo.

Funções

Algumas funções foram criadas, entre elas:

void fetch(void);

Lê uma instrução da memória e a coloca em ri, atualizando o pc para apontar para a próxima instrução (somando 4).

void decode (void);

Extrai 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

A extração se dá por meio de máscaras nos bits da instrução e deslocamentos para acessar a área correta de cada campo.

void execute (void);

Executa a instrução que foi lida pela função fetch() e decodificada por decode(). É a parte em que a instrução é implementada de fato, por meio de um switch do opcode e do funct.

void step(void);

Simula uma opção do MARS, executando apenas uma instrução do MIPS:

step() => fetch(), decode(), execute()

Após ser executada, o usuário tem a opção de ver o conteúdo da memória e do banco de registradores.

void run(void);

Simula uma opção do MARS, executando o programa até encontrar uma chamada de sistema para encerramento, ou até o pc ultrapassar o limite do segmento de código (2k words).

Após ser executada, o usuário pode de ver o conteúdo da memória e do banco de registradores.

void dump_mem(int start, int end, char format);

Permite visualizar o conteúdo da memória, imprimindo-o a partir do endereço start até o endereço end. O formato pode ser em hexa (‘h’) ou decimal (‘d’).

void dump_reg(char format);

Permite visualizar o conteúdo dos registradores, imprimindo o banco de registradores e

os registradores pc, hi e lo. O formato pode ser em hexa (‘h’) ou decimal (‘d’).

As funções de acesso à memória são:

int32_t lb(uint32_t *address, uint16_t kte);

int32_t lh(uint32_t *address, uint16_t kte);

int32_t lw(uint32_t *address, uint16_t kte);

void sb(uint32_t *address, uint16_t kte, int8_t dado);

void sh(uint32_t *address, uint16_t kte, int16_t dado);

void sw(uint32_t *address, uint16_t kte, int32_t dado);

Os endereços devem ser escritos na forma de byte. A operação de leitura retorna um inteiro com o byte , ou meia-palavra, ou palavra lidas na posição menos significativa. A escrita de um termo

deve colocá-lo na posição correta dentro da palavra de memória. O ponteiro *address aponta para o registrador que fornece a valor base da memória para o endereçamento.

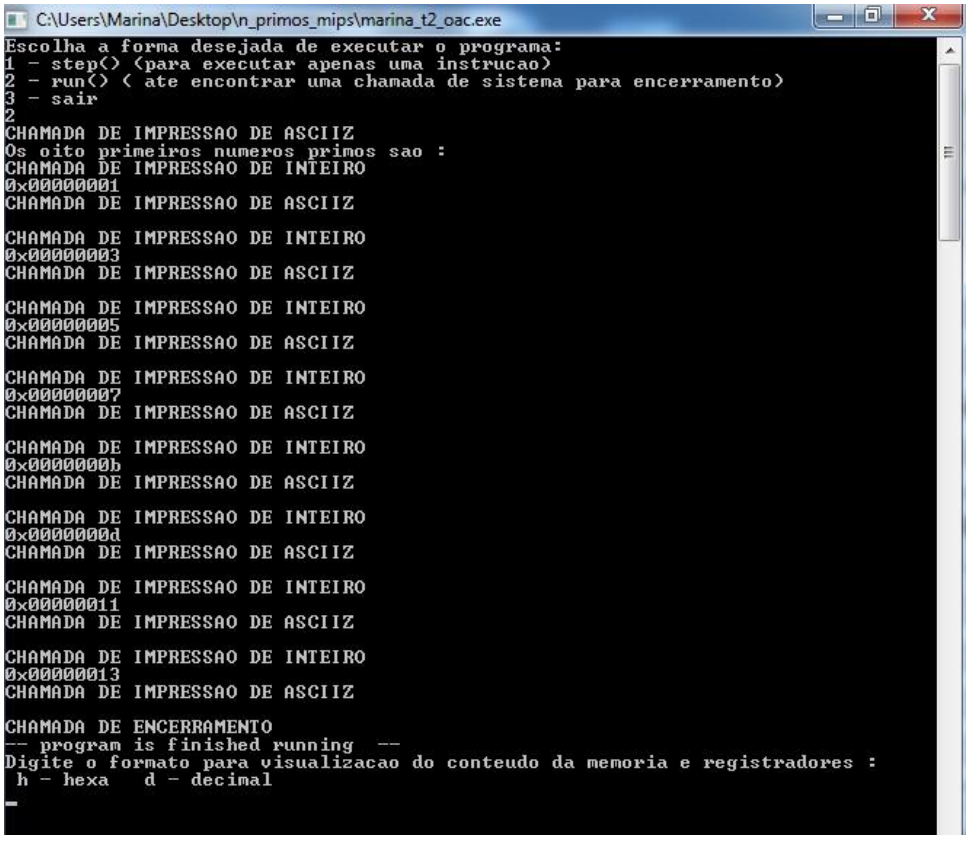
Testes e resultados

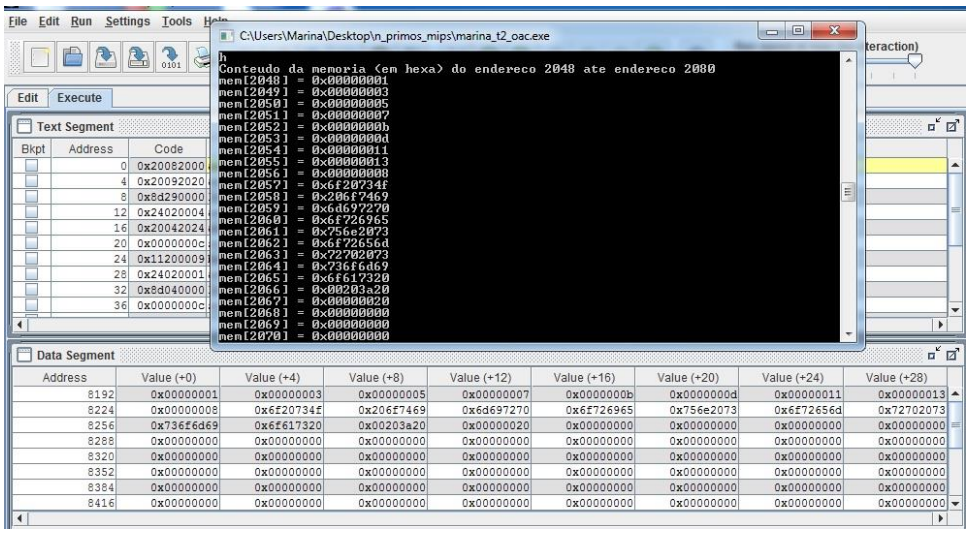
Alguns programas foram testados, entre eles

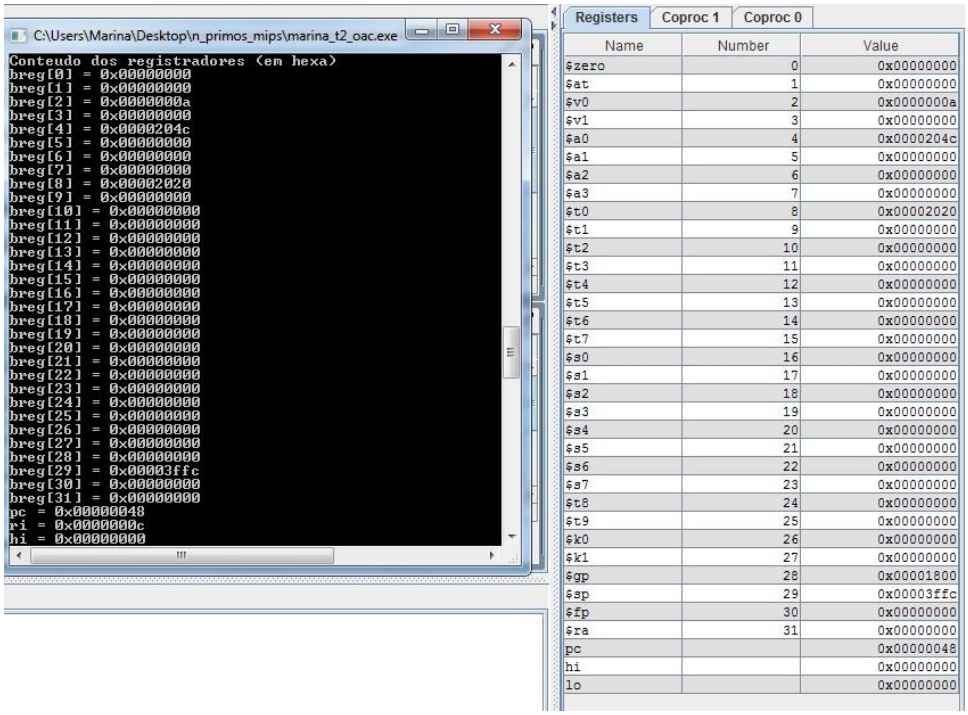
1 - mips.asm - programa da primeira página da descrição do trabalho - imprime os oitos primeiros números primos.

Neste programa, a simulação correspondeu corretamente a todas as instruções, e a saída para o usuário, assim como o conteúdo da memória e dos registradores , corresponderam à saída do MARS.

Saída:



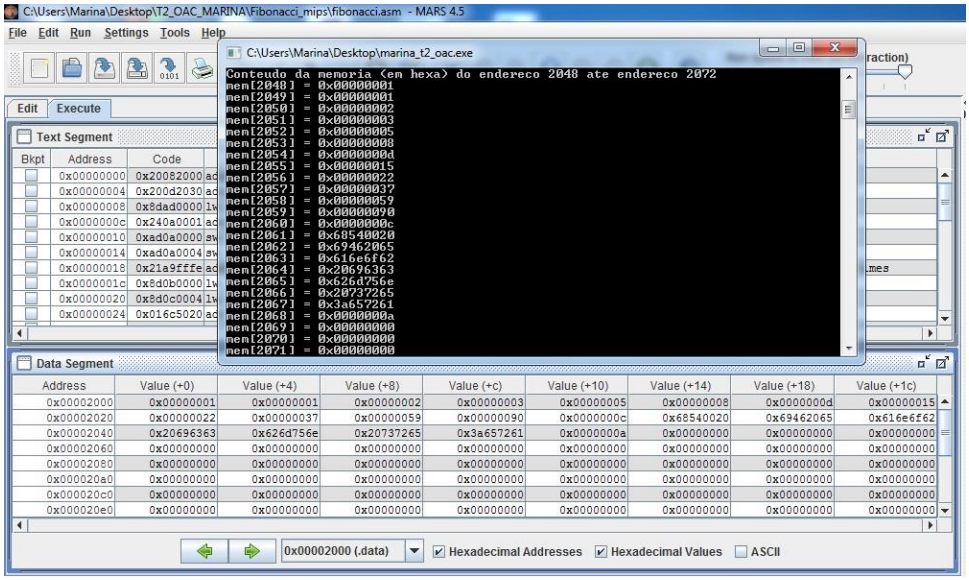


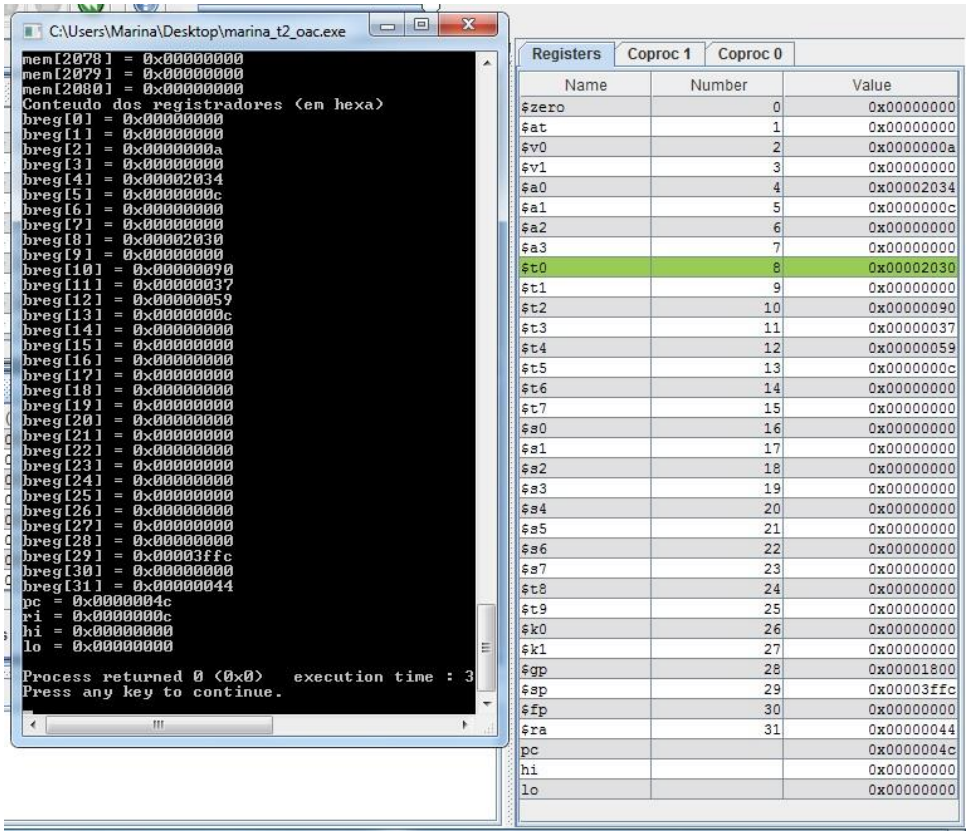


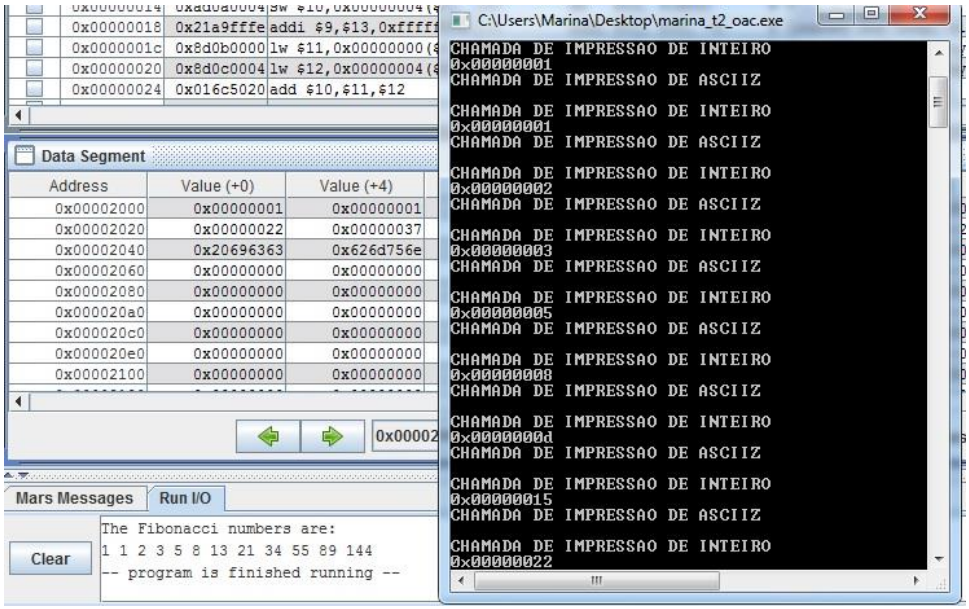
2 - Fibonacci.asm - programa que acompanha o MARS - imprime alguns números da série de Fibonacci.

Neste programa, a simulação correspondeu corretamente a todas as instruções, e a saída para o usuário, assim como o conteúdo da memória e dos registradores , corresponderam à saída do MARS.

Saída:





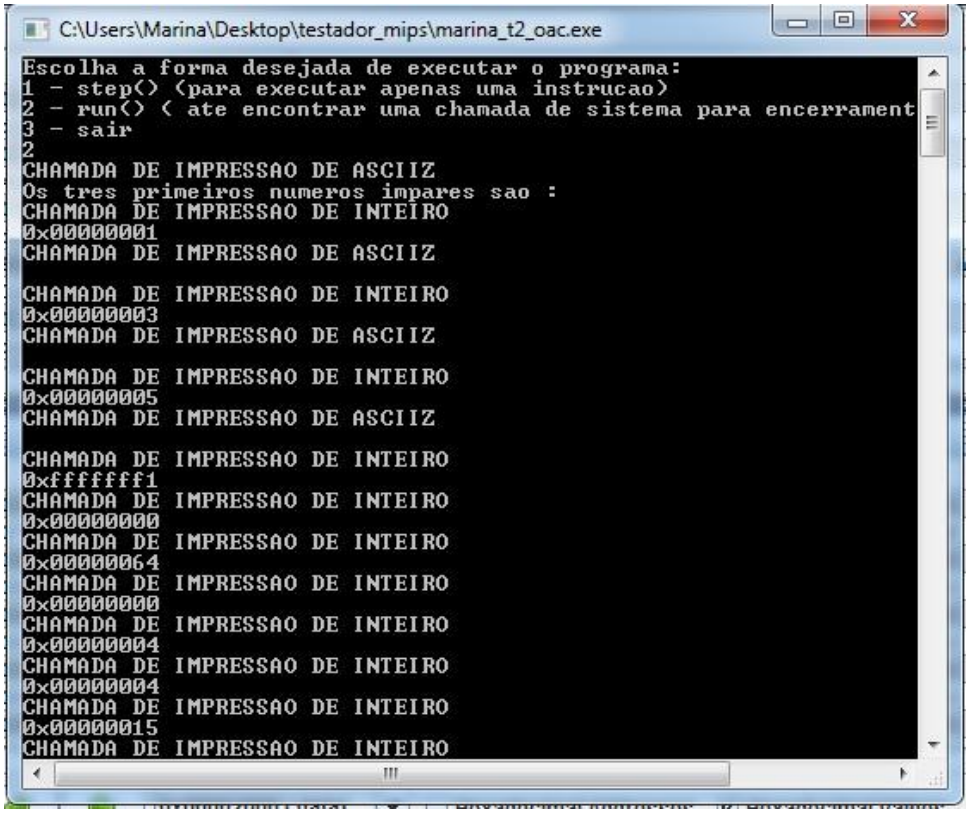


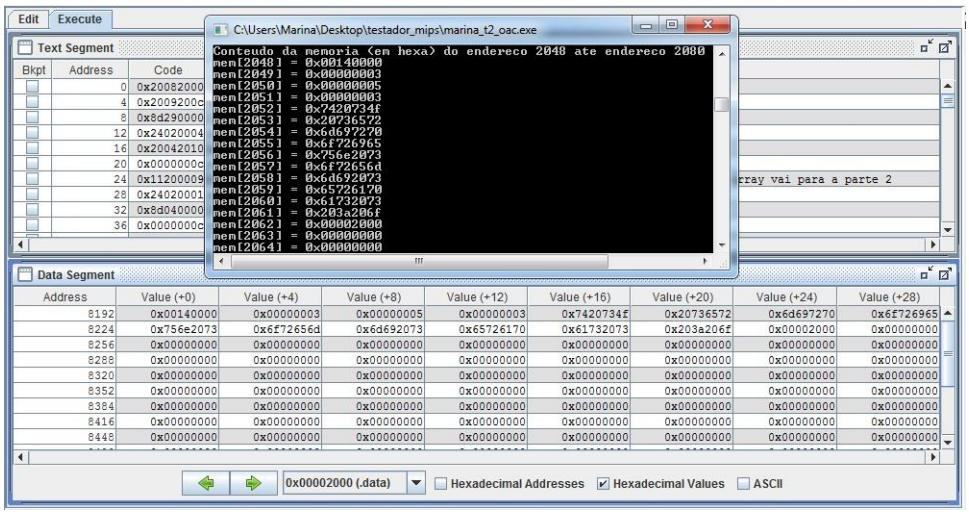
3 - testador.asm - programa que testa todas as instruções implementadas pelo simulador.

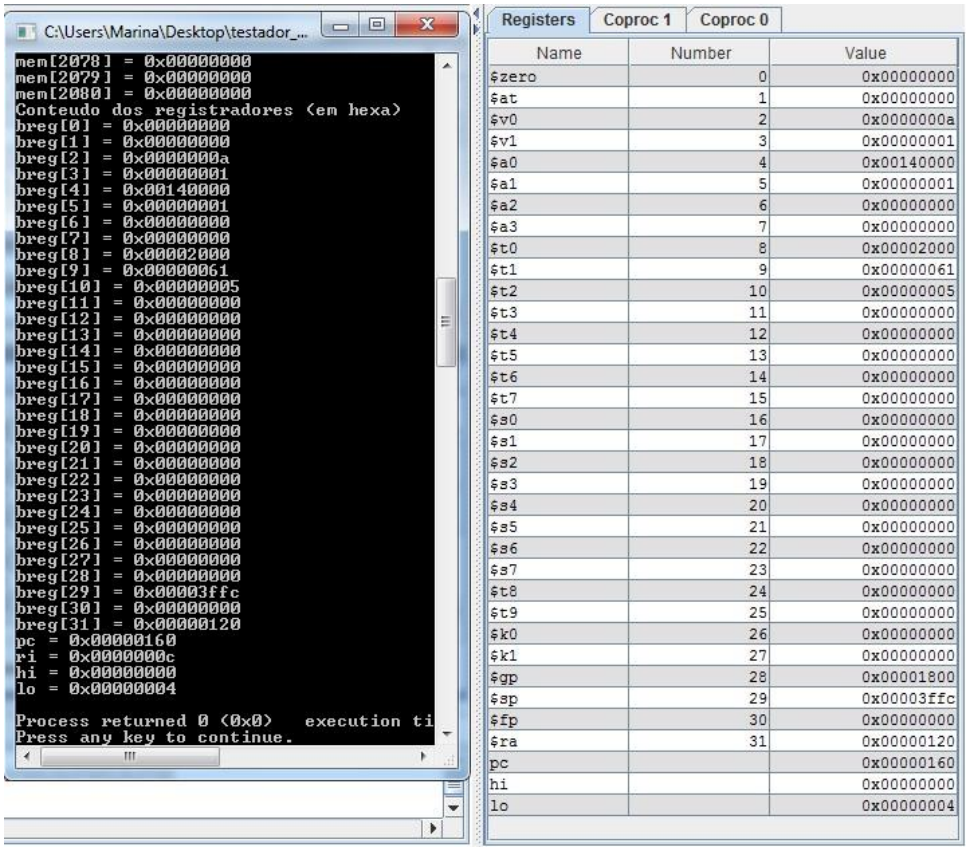
Neste programa, a simulação correspondeu corretamente a todas as instruções, e a saída para o usuário, assim como o conteúdo da memória e dos registradores , corresponderam à saída do MARS.

A maioria dos resultados das instruções é impressa na tela de saída para o usuário, mas não todas. Porém, se o usuário deseja visualizar cada resultado, basta selecionar a opção step() para tal instrução e aceitar um dump_mem e dump_reg.

Saída


C:\Users\Marina\Desktop\testador_mips\marina_t2_oac.exe
Escolha a forma desejada de executar o programa:
1 - step() (para executar apenas uma instrucao)
2 - run() (ate encontrar uma chamada de sistema para encerrament
3 - sair
2
CHAMADA DE IMPRESSAO DE ASCIIZ
Os tres primeiros numeros impares sao :
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000001
CHAMADA DE IMPRESSAO DE ASCIIZ
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000003
CHAMADA DE IMPRESSAO DE ASCIIZ
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000005
CHAMADA DE IMPRESSAO DE ASCIIZ
CHAMADA DE IMPRESSAO DE INTEIRO
0xfffffff1
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000000
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000064
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000000
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000004
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000004
CHAMADA DE IMPRESSAO DE INTEIRO
0x00000015
CHAMADA DE IMPRESSAO DE INTEIRO


Edit
Execute
Text Segment
Bkpt
Address
Code
C:\Users\Marina\Desktop\testador_mips\marina_t2_oac.exe
Conteudo da memoria (em hexa) do endereco 2048 ate endereco 2080
mem[2048] = 0x00140000
mem[2049] = 0x00000003
mem[2050] = 0x00000005
mem[2051] = 0x00000003
mem[2052] = 0x7420734f
mem[2053] = 0x20736572
mem[2054] = 0x6d697270
mem[2055] = 0x6f726965
mem[2056] = 0x756e2073
mem[2057] = 0x6f72656d
mem[2058] = 0x6d692073
mem[2059] = 0x65726170
mem[2060] = 0x61732073
mem[2061] = 0x203a206f
mem[2062] = 0x00002000
mem[2063] = 0x00000000
mem[2064] = 0x00000000
rray vai para a parte 2
Data Segment
Address
Value (+0)
Value (+4)
Value (+8)
Value (+12)
Value (+16)
Value (+20)
Value (+24)
Value (+28)
0x00002000 (.data)
Hexadecimal Addresses
Hexadecimal Values
ASCII


C:\Users\Marina\Desktop\testador_...
mem[2078] = 0x00000000
mem[2079] = 0x00000000
mem[2080] = 0x00000000
Conteudo dos registradores (em hexa)
breg[0] = 0x00000000
breg[1] = 0x00000000
breg[2] = 0x0000000a
breg[3] = 0x00000001
breg[4] = 0x00140000
breg[5] = 0x00000001
breg[6] = 0x00000000
breg[7] = 0x00000000
breg[8] = 0x00002000
breg[9] = 0x00000061
breg[10] = 0x00000005
breg[11] = 0x00000000
breg[12] = 0x00000000
breg[13] = 0x00000000
breg[14] = 0x00000000
breg[15] = 0x00000000
breg[16] = 0x00000000
breg[17] = 0x00000000
breg[18] = 0x00000000
breg[19] = 0x00000000
breg[20] = 0x00000000
breg[21] = 0x00000000
breg[22] = 0x00000000
breg[23] = 0x00000000
breg[24] = 0x00000000
breg[25] = 0x00000000
breg[26] = 0x00000000
breg[27] = 0x00000000
breg[28] = 0x00000000
breg[29] = 0x00003ffc
breg[30] = 0x00000000
breg[31] = 0x00000120
pc = 0x00000160
ri = 0x0000000c
hi = 0x00000000
lo = 0x00000004
Process returned 0 (0x0) execution ti
Press any key to continue.
Registers
Coproc 1
Coproc 0
Name
Number
Value
$zero 0 0x00000000
$at 1 0x00000000
$v0 2 0x0000000a
$v1 3 0x00000001
$a0 4 0x00140000
$a1 5 0x00000001
$a2 6 0x00000000
$a3 7 0x00000000
$t0 8 0x00002000
$t1 9 0x00000061
$t2 10 0x00000005
$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
$t8 24 0x00000000
$t9 25 0x00000000
$k0 26 0x00000000
$k1 27 0x00000000
$gp 28 0x00001800
$sp 29 0x00003ffc
$fp 30 0x00000000
$ra 31 0x00000120
pc 0x00000160
hi 0x00000000
lo 0x00000004