

Trabalho Prático 5

Introdução ao *Assembly* do MIPS

Objetivos


- Introdução ao simulador MARS
- Tradução dum programa em C para *Assembly*
- Execução e *Debug* dum programa *Assembly*

Guião

1. Panorâmica geral do simulador MARS

O simulador MARS (MIPS Asembler and Runtime Simulator) é um *software* gratuito que permite editar e executar programas em *Assembly*. Essencialmente, é composto por um editor sensível à sintaxe (*syntax highlight*) e por um simulador que permite a execução e *debug*. Apresentamos abaixo uma breve descrição destas duas componentes.

1.1 Janela de edição

A Figura 1 apresenta o aspeto da janela do editor, onde se podem destacar as secções de dados (*.data*) e de código (*.text*) e os comentários (verde). Na secção de código, podemos identificar claramente as mnemónicas *Assembly* (azul), os registos (vermelho) e os *labels* (preto), graças à utilização de diferentes cores. O programa pode ser *assemblado* carregando no botão . Caso o programa não contenha erros de sintaxe, o MARS muda para a janela de execução e de *debug* da Figura 2.

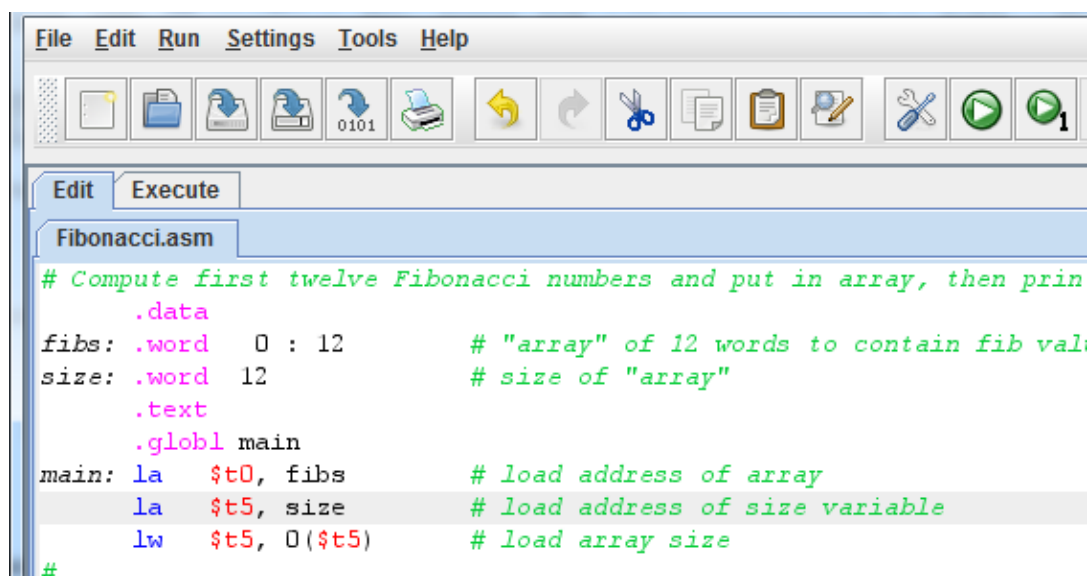




Figura 1 - Janela de edição

1.2 Janela de execução

Nesta janela são apresentados os segmentos de código (.text) e de dados (.data) do programa *Assembly*. No painel do lado direito temos o conjunto dos 32 registos do MIPS, cujo valor pode também ser editado. A consola (*Run I/O*), na parte inferior, permite ao programa interagir com o utilizador, usando chamadas-ao-sistema (*syscalls*) específicas para esse efeito, por exemplo, *print_string*. A execução do programa pode ser feita duma só vez (comando *run* ) , ou em modo passo-a-passo (comando *single step* ). É ainda possível a introdução de pontos-de-paragem (*breakpoints*) para facilitar a deteção e correção de erros (*debug*).

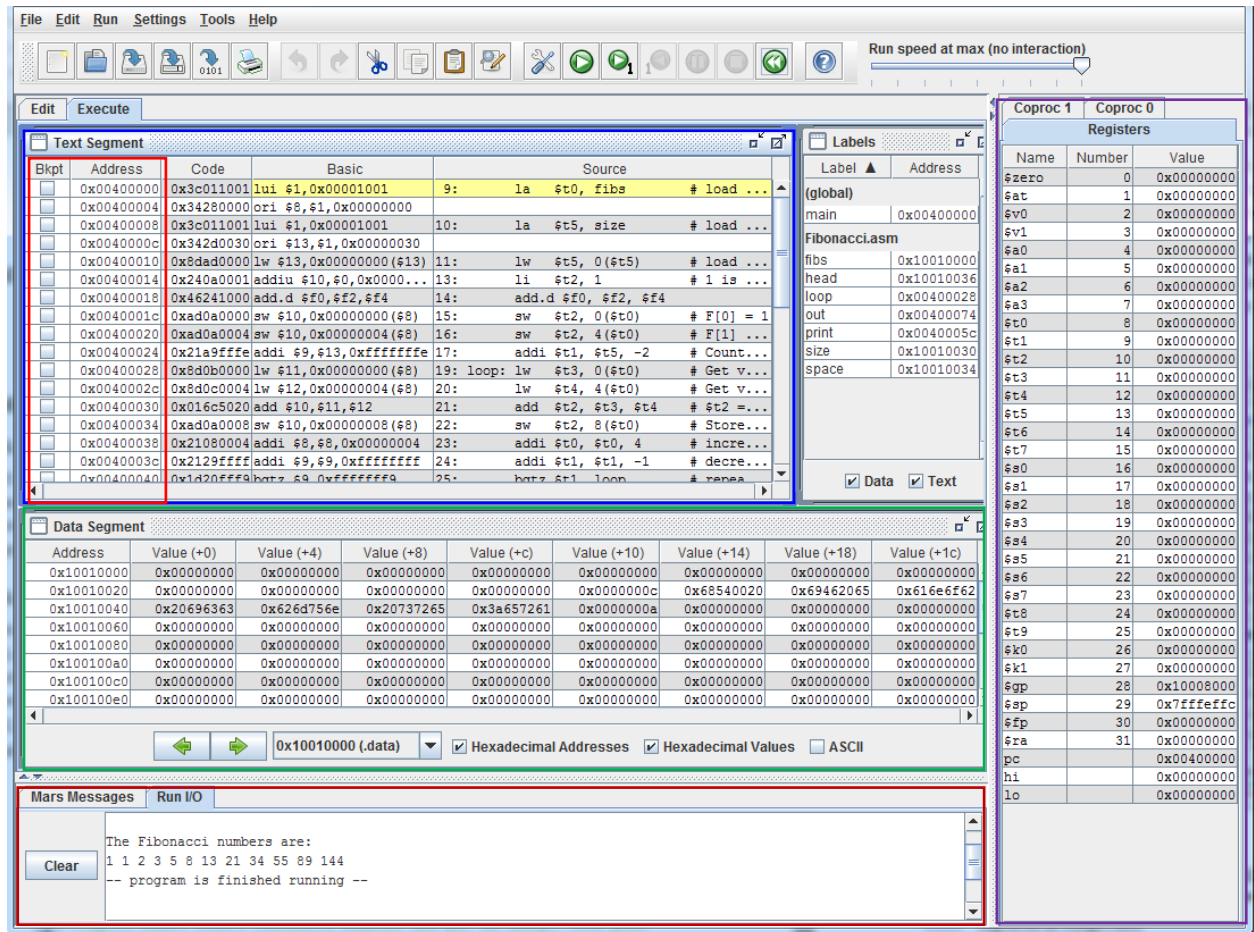


Figura 2 - Janela de execução

- Segmento de código (.text)
- Segmento de dados (.data)
- Registos
- Consola (Run I/O)
- Pontos-de-paragem (Breakpoints)

2. Programação em *Assembly* no MARS

Edite o programa *Assembly* apresentado no Anexo C e guarde-o num ficheiro com a extensão ".asm" (e.g., "trab5_1.asm"). Faça a *assemblagem* do código que editou.

2.1. Segmento de código

Observe o conteúdo do painel *text segment*, em particular a primeira e a última colunas.

- A partir de que endereço foi colocada a 1ª instrução *Assembly* que escreveu?
- Quais as instruções da máquina real que correspondem a essa instrução?
- Qual o código máquina de cada uma das instruções da máquina real que identificou na alínea anterior?
- Qual o valor atual do registo *Program Counter*?

2.2. Segmento de dados

Observe o conteúdo do painel *data segment*. Em que endereços estão colocadas as *strings*: *prompt* e *result*? Se necessário consulte a tabela ASCII de codificação de caracteres.

2.3. Execução e teste

- Run:** Execute o programa e verifique se funciona corretamente.
- Single-Step:** Execute novamente o programa, mas agora passo a passo.
- BreakPoints:** Introduza um *breakpoint* no endereço correspondente à instrução "move \$t0, \$v0", e execute novamente o programa. Execute a parte restante do programa passo a passo, verificando o resultado de todas as instruções. O resultado da execução pode ser visualizado no painel de registos, no segmento de dados e ainda na consola (*Run I/O*).

3. Anexos

3.1 Anexo A - Programa em Java

```
// Programa em linguagem Java

import java.io.*;
import java.util.Scanner;

public class aula7
{
    public static void main(String[] args) throws NumberFormatException
    {
        String result = "\nO numero em que pensaste e': ";
        String prompt = "1. Pensa num numero!\n2. Adiciona 3\n
                        3. Multiplica o resultado por 2\n
                        4. Subtrai o numero em que pensaste\n\n\t
                        Qual o resultado? ";

        int num;

        System.out.println( prompt );
        num = new Scanner( System.in ).nextInt();
        System.out.print(result);
        System.out.println(num-6);

        // print_str( prompt );
        // num = read_int();
        // print_str( result );
        // print_int( num - 6 );
    }
}
```

3.2 Anexo B - Programa em C

/* Programa correspondente em linguagem C */

```
void main(void)
{
    char result[] = "\n0 numero em que pensaste e': ";
    char prompt[] = "1. Pensa num numero!\n2. Adiciona 3\n"
                   "3. Multiplica o resultado por 2\n"
                   "4. Subtrai o numero em que pensaste\n"
                   "\n\tQual o resultado? ";

    int num;
    print_str( prompt );
    num = read_int();
    print_str( result );
    print_int( num - 6 );

    exit();
}
```

3.3 Anexo C - Programa em Assembly

```
# Tradução do programa em linguagem C para assembly do MIPS

.data
result: .asciiz "\n0 numero em que pensaste e': "
prompt: .ascii "1. Pensa num numero!\n"
        .ascii "2. Adiciona 3\n"
        .ascii "3. Multiplica o resultado por 2\n"
        .ascii "4. Subtrai o numero em que pensaste\n"
        .asciiz "\n\tQual o resultado? "
#####
.text
.globl main
# int num;      "num" reside no registo $t0
#
main:         la      $a0, prompt      # $a0 = prompt
              li      $v0, 4           # $v0 = 4 (syscall "print_str")
              syscall                      # print_str( prompt )
              #
              li      $v0, 5           # $v0 = 5 (syscall "read_int")
              syscall                      # read_int() (o valor lido é
              #                               # devolvido no reg. $v0)
              move    $t0, $v0         # $t0 = $v0 ( num = read_int() )
              #
              la      $a0, result      # $a0 = result
              li      $v0, 4           # $v0 = 4 (syscall "print_str")
              syscall                      # print_str( result )
              #
              sub     $a0, $t0, 6       # $a0 = $t0 - 6 ( $a0 = num - 6 )
              li      $v0, 1           # $v0 = 1 (syscall "print_int")
              syscall                      # print_int( num - 6 )
              #
              li      $v0, 10          #
              syscall                      # exit()
```