

LSDI 2015/16 - Trabalho laboratorial 6

Análise de um programa em *assembly* MIPS com MARS

1. Introdução

Este trabalho consiste na análise de um programa em *assembly* MIPS com recurso à ferramenta de simulação MARS (<http://courses.missouristate.edu/kenvollmar/mars/>). A iniciação ao uso desta ferramenta é o principal objetivo do trabalho, permitindo a familiarização com MARS necessária à posterior utilização no desenvolvimento de programas em linguagem *assembly*.

2. Programa a analisar

Considere um programa que calcula o maior elemento de uma sequência (vetor) de números inteiros armazenada em memória. O código *assembly* seguinte (disponível na pasta de Conteúdos do Sigarra) é uma possível solução, na qual se inclui um vetor para se proceder à respetiva simulação com MARS. Analise o código atentamente com a ajuda dos comentários fornecidos.

```
.data
dim:      .word 7                # Número de elementos do vetor
V:        .word 3, -5, 21, 2, -10, 50, 9  # Vetor

.text
la        $s0, V                # Inicializa registos para
lw        $t0, dim              # percorrer vetor
lw        $t1, 0($s0)           # Máximo inicial
ciclo:    addi   $t0, $t0, -1     # Atualiza número de elementos a percorrer
          beq    $t0, $zero, escreve # Se percorreu todos, escreve e termina
          addi   $s0, $s0, 4      # Atualiza endereço para ler próximo elemento
          lw     $t2, 0($s0)      # Elemento atual
          slt    $t3, $t1, $t2    # $t3=1 indica que foi encontrado novo máximo
          beq    $t3, $zero, salta
          add    $t1, $t2, $zero  # Atualiza máximo
salta:    j      ciclo
escreve:  add    $a0, $t1, $zero  # Passa para $a0 o inteiro a escrever
          addi   $v0, $zero, 1    # Identifica a operação 'print integer'
          syscall                # Chamada ao sistema (executa 'print integer')
          addi   $v0, $zero, 10   # Termina
          syscall
```

3. Desenvolvimento

- Após iniciar o MARS defina o modelo de memória em que o código máquina de um programa ocupa endereços de memória a partir de 0. Para tal, escolha *Settings* → *Memory Configuration ...* → *Compact, Text at Address 0* → *Apply and Close*. Neste modo, os dados declarados com a diretiva *.data* são armazenados a partir do endereço 0x00002000.
- Edite o código do programa e grave-o num ficheiro com extensão *.asm*. Faça *Assemble* e torne visível a janela *Labels*, selecionando a opção *Symbol table* em *Settings*. O código máquina resultante é visível na janela *Text segment* e os dados em memória são mostrados na janela *Data segment*.
- Para simular a execução do programa pode fazê-lo de dois modos:
 - *Run* → *Go*, executa o programa completo;
 - *Run* → *Step*, executa uma instrução, de cada vez que este modo é invocado.
 O modo *Step* é o mais útil quando se testa o funcionamento de um fragmento de código, permitindo ver os efeitos decorrentes da instrução executada. Em alternativa, no modo *Go* podem definir-se pontos de paragem (*breakpoints*) em instruções “estratégicas”, onde se pretenda observar o estado de registos e/ou memória, facilitando a depuração do programa. A inserção/remoção de pontos de paragem faz-se na janela *Text Segment* com o botão existente à esquerda do endereço de cada instrução.

Responda às questões que a seguir se apresentam.

a) Que significado atribui à diretiva `.text`?

R:

b) Indique o endereço de memória onde começa o código máquina.

R:

c) Qual a instrução no endereço `0x00000010`?

R:

d) Indique o código da instrução `addi $t0, $t0, -1`.

R:

e) Indique o código da instrução `j ciclo`.

R:

f) A instrução `beq $t0, $zero, escreve` foi convertida em `beq $8, $0, 0x00000006` pelo *assembler*. Explique o significado e o valor do terceiro argumento desta instrução.

R:

g) Indique o valor do registo `PC` antes de simular o programa.

R:

h) Indique o endereço de memória onde começa o armazenamento de dados.

R:

i) Indique o endereço base do vetor `v` em memória.

R:

j) Quantos bytes ocupa o vetor `v` em memória?

R:

k) No endereço `0x00002014` encontra o valor `0xffffffff6`. Que significado lhe atribui?

R:

l) Que endereços ocupa o último elemento de `V`?

R:

m) Se trocar a ordem das linhas que contêm a diretiva `.word`, em que posição de memória fica o valor de `dim`?

R:

n) Qual o significado do valor de `$s0` após a execução da pseudo-instrução `la $s0, v`?

R:

o) Qual o endereço correspondente a `escreve`? O conteúdo nesse endereço representa uma instrução ou dados do programa?

R:

p) Qual o significado da instrução `addi $v0, $zero, 1` que precede `syscall`?

R:

q) Quantas vezes é executada a instrução `add $t1, $t2, $zero`?

R:

r) Indique o resultado da execução do programa.

R:

s) Repita a alínea anterior após alterar em memória o valor de `dim` para 3.

R:

t) Se substituir a instrução `slt $t3, $t1, $t2` por `slt $t3, $t2, $t1`, que funcionalidade adquire o programa?

R: