**Laboratório II - A máquina Virtual MARS e as Variáveis de Programa**

Nome: Gabriel Gatti da Silva Matrícula: 2021100336

Nome: Thiago F. N. Lahass Matrícula: 2021100178

Data: 24/10/22

**1. OBJETIVOS**

Verificar como as instruções sintéticas (ou pseudoinstruções) MIPS podem ser reconhecidas no código por meio do emulador MARS;

Compreender como as instruções sintéticas se expandem para sequências de instruções em linguagem de máquina;

Entender como o MIPS endereça a memória de dados.

**2. ATIVIDADES**

**2.1. Usando instruções sintéticas ou pseudoinstruções**

Identificar as instruções sintéticas no código fonte e preencher a tabela a seguir ( a variável X foi substituída pelo inteiro 100).

| **Endereço** | **Instrução sintética** | **Instruções nativas** | **Efeito** |
| --- | --- | --- | --- |
| 0x00400000 | la $ra, main | lui $1, 0x00000040  ori $31, $1, 0x00000000 | $1 = 0x00400000  $31 = $1 or 0x00000000 |
| 0x0040000c | li $t0, 100 | addiu $8, $0, 0x00000064 | $8 = $0 + 0x00000064 |
| 0x00400010 | move $t1, $t0 | addu $9, $0, $8 | $9 = $0 + $8 |
| 0x00400014 | la $t2, var2 | lui $1, 0x00001001  ori $10, $1, 0x00000004 | $1 = 0x10010000  $10 = $1 or 0x00000004 |
|  |  |  |  |
| 0x0040001c | lw $t3, var2 | lui $t1, 0x00001001  lw $11, 0x00000004($1) | $1 = 0x10010000  $11 = conteúdo do endereço ( 0x10010004 ) |
| 0x00400024 | sw $t2, var1 | lui $t1, 0x00001001  sw $10, 0x00000000($1) | $1 = 0x10010000  conteúdo do endereço  ( 0x10010000 ) = $10 |

Tabela 1 - Instruções sintéticas e suas instruções nativas equivalentes e efeitos

**2.2. Carregando um endereço em um registrador**

Na tabela abaixo, insira a constante que você encontra com os primeiros lui e ori em seu programa, em formato hexadecimal. Explique o que estas constantes representam no seu programa.

| Instrução | Constante | Significado da constante |
| --- | --- | --- |
| lui $1, 0x00001001 | 0x00001001 | Parte mais significativa do endereço a ser carregado para o registrador de destino |
| ori $10, $1, 0x00000004 | 0x00000004 | Parte menos significativa do endereço a ser carregado para o registrador de destino |

Tabela 2 - Constantes e seu significados nas instruções lui e ori

**2.3. Endereçamento em MIPS**

-- Q1: Criar um programa em assembly MIPS que tenha as seguintes características:

* Reserve espaço na memória para quatro variáveis chamadas var1 a var4 do tamanho da palavra.
* Defina valores iniciais para estas variáveis
* Reserve espaço na memória para duas variáveis chamadas “primeiro” e “último” do tipo Byte. O valor inicial de primeiro deve ser a primeira letra do seu primeiro nome e o valor inicial do último deve ser a primeira letra do seu último nome.
* O programa troca os valores das variáveis na memória: o novo valor de var1 será o valor inicial de var4, o novo valor de var2 será o valor inicial de var3, var3 receberá o valor inicial de var2, e finalmente o var4 obterá o valor inicial de var1.

**Solução:**

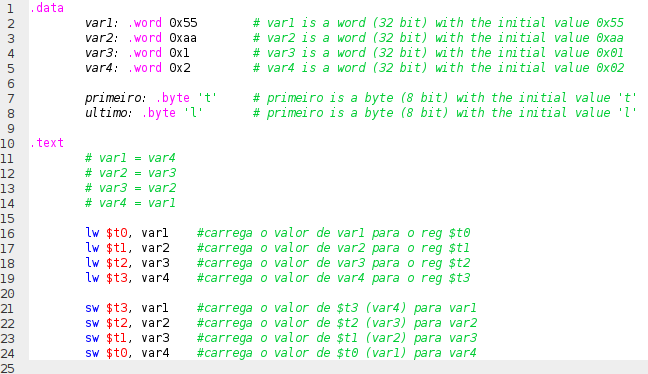


Imagem 1 - Código para a questão Q1

-- Q2: Encontre manualmente (faça e mostre suas contas) o valor do deslocamento usado para determinar o endereço de cada variável no seu programa desde o início do segmento de dados. O deslocamento será a distância em Bytes entre o início do segmento de dados e o local onde a variável é armazenada. Em seguida, compare estes valores de deslocamento com os que aparecem na tabela de labels do seu programa no MARS. Use as chamadas de sistema (syscall) para verificar o que está armazenado na memória no segmento de dados (basta imprimir o que está na memória, comparando com os valores armazenados na tabela “Data Segment” do MARS). Apresente os resultados e comparações entre o valor esperado e o valor real.

**Solução:**

Considerando que o segmento de dados começa no endereço 0x10010000, temos:

* A variável var1 é a primeira do nosso programa, ou seja, seu endereço é o primeiro endereço disponível no nosso segmento de dados, isto é, 0x10010000.
* A próxima variável é a var2, como a primeira variável é do tipo word, ou seja, possui 4 bytes, então a nossa segunda variável está no endereço inicial (0x10010000) somado a um deslocamento de 4 bytes ( espaço ocupado pela variável var1). Portanto o endereço de var2 é 0x10010000 + 0x00000004, ou seja, 0x10010004.
* Como a variável var2 também é do tipo word, então também ocupa um espaço de 4 bytes. Logo o endereço de var3 é o endereço inicial do segmento de dados (0x10010000) deslocado em 8 bytes, ou seja, 0x10010008.
* Da mesma forma, o endereço de var4 está em 0x10010000 deslocado de 12 bytes, ou seja, 0x1001000c.
* Já as variáveis ‘primeiro’ e ‘ultimo’ ocupam somente um único byte, e portanto o endereço de ‘primeiro’ é o endereço inicial do segmento de dados (0x10010000) deslocado em 16 bytes, ou seja, 0x10010010, e o endereço de ‘ultimo’ é o endereço inicial do segmento de dados (0x10010000) deslocado em 17 bytes, ou seja, 0x10010011.

Esses resultados podem ser vistos também na imagem abaixo, como esperado:

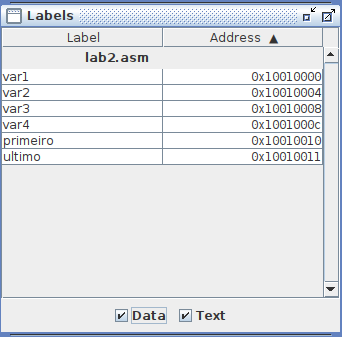


Imagem 2 - Variáveis e seus endereços no programa da Q2

Além disso, podemos que ao fazer a chamada do syscall para as variáveis e se pode ver que esses dados correspondem ao que está no segmento de dados, como é possível verificar nas imagens abaixo:

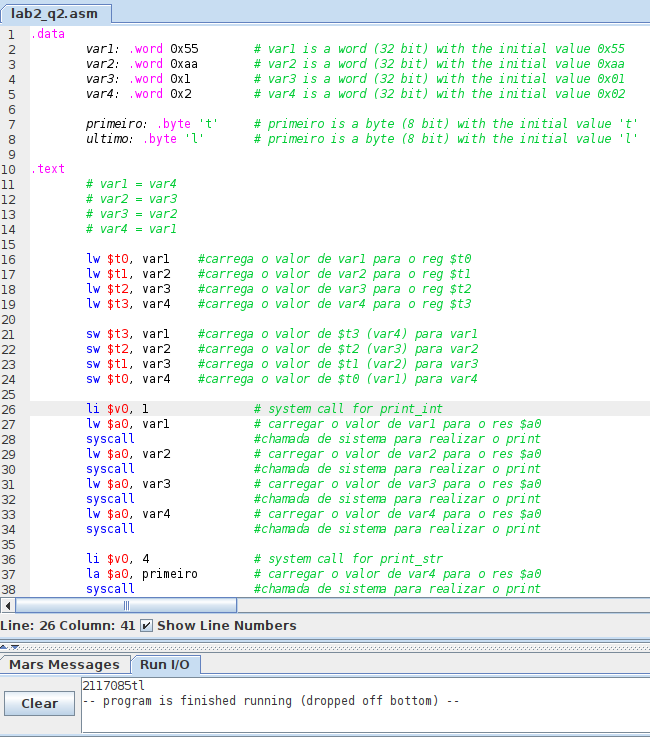


Imagem 3 - Código do programa Q2 e sua respectiva saída logo abaixo

Vemos que a saída corresponde aos dados na memória abaixo:

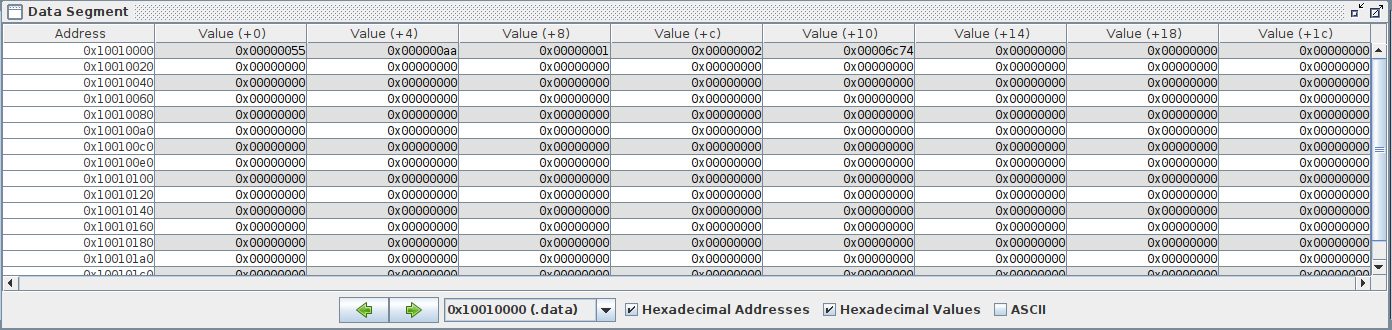


Imagem 4 - Data Segment

-- Q3: Quantas instruções assembly e quantas instruções de máquina foram utilizadas no seu programa? Como você determina o tamanho do programa, em Bytes, a partir destas informações?

**Solução:**

Como é possível ver na imagem 3, foram utilizadas 20 instruções assembly. Já as instruções de máquina podem ser vistas na imagem abaixo:

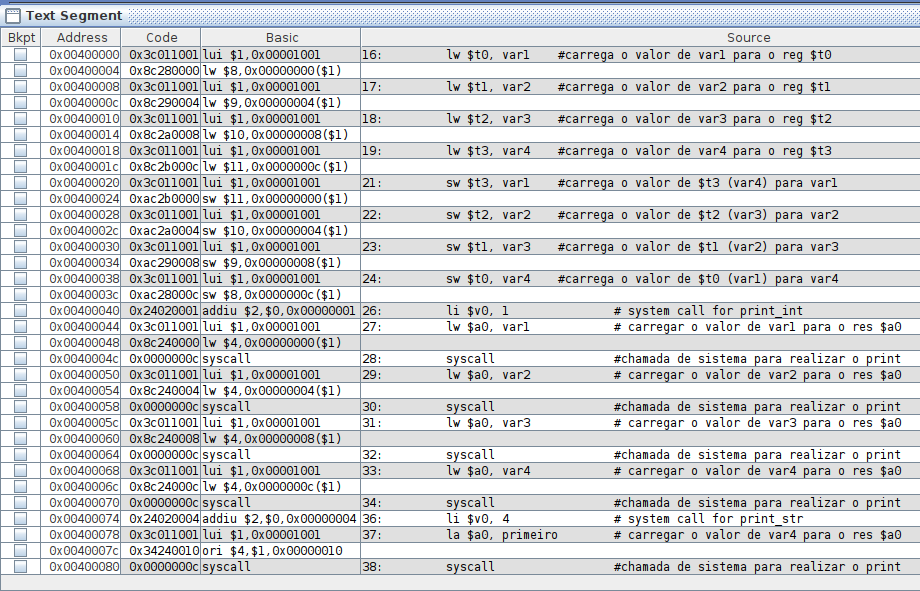


Imagem 5 - Text Segment

Onde podemos ver que há 33 instruções de máquina.

Cada instrução de máquina é uma *word*, ou seja, possui 4 bytes. Portanto é possível determinar o tamanho do programa contado o número de instruções de máquina que ele contém, além do número de variáveis que há nele. No exemplo acima, temos 33 instruções, ou seja, elas ocupam 33 \* 4 = 132 bytes no segmento de texto. Além disso, temos 4 variáveis inteiras, que ocupam 4 bytes cada, e outras duas variáveis do tipo byte, que ocupam um byte cada, porém a última word é totalmente ocupada por questões de arquitetura. Finalmente, o nosso programa ocupa um total de 132 + 16 + 4 = 152 bytes.