

Introdução ao MARS

O MARS (MIPS *Assembly and Runtime Simulator*) é um programa simulador do processador MIPS 32 bits desenvolvido para fins educacionais. É uma máquina virtual, isto é, um programa que interpreta a arquitetura de um determinado processo (no caso o MIPS) e a executa em uma arquitetura distinta (como o IA-32, seu computador). O MARS fornece uma interface para o modelo de visualização do MIPS (*Programmer's View*), que consiste essencialmente em suas instruções, registradores e mecanismos de acesso à memória. Também, apresenta outras ferramentas de análise, como o simulador de cache de dados (*Data Cache Simulator*). O simulador MARS pode ser obtido acessando o endereço <http://courses.missouristate.edu/kenvollmar/mars/>, foi desenvolvido na linguagem Java requerendo o Java J2SE 1.5 (ou superior) SDK.

O MARS (versão 4.5) possui dois modos de visualização: **Edição e Execução**.

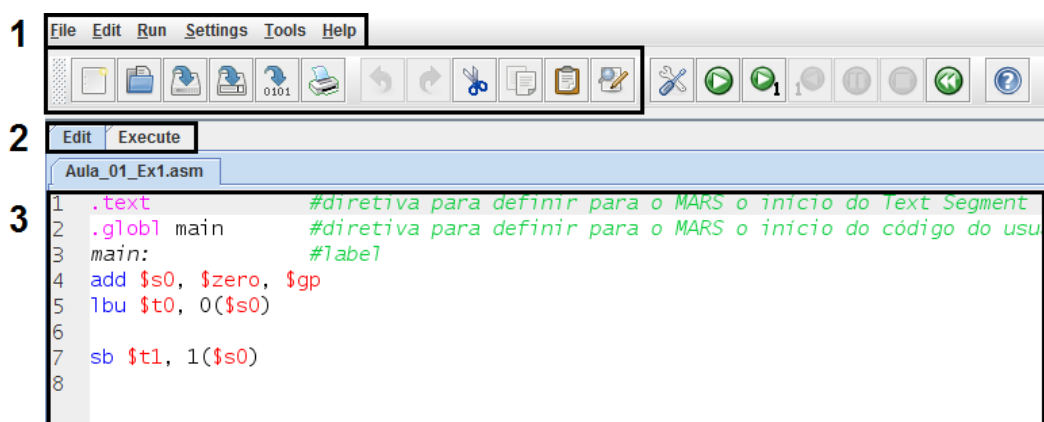


Fig. 1 – Janela de Edição do MARS.

1 – Menus e Atalhos: principais menus e atalhos que possibilitam a execução de operações (por exemplo: abertura e criação de arquivos; função copiar e colar) conforme o processo que está sendo executado.

2 – Modos de visualização: Edição ou Execução.

3 – Editor: para a programação da linguagem de montagem do MIPS.

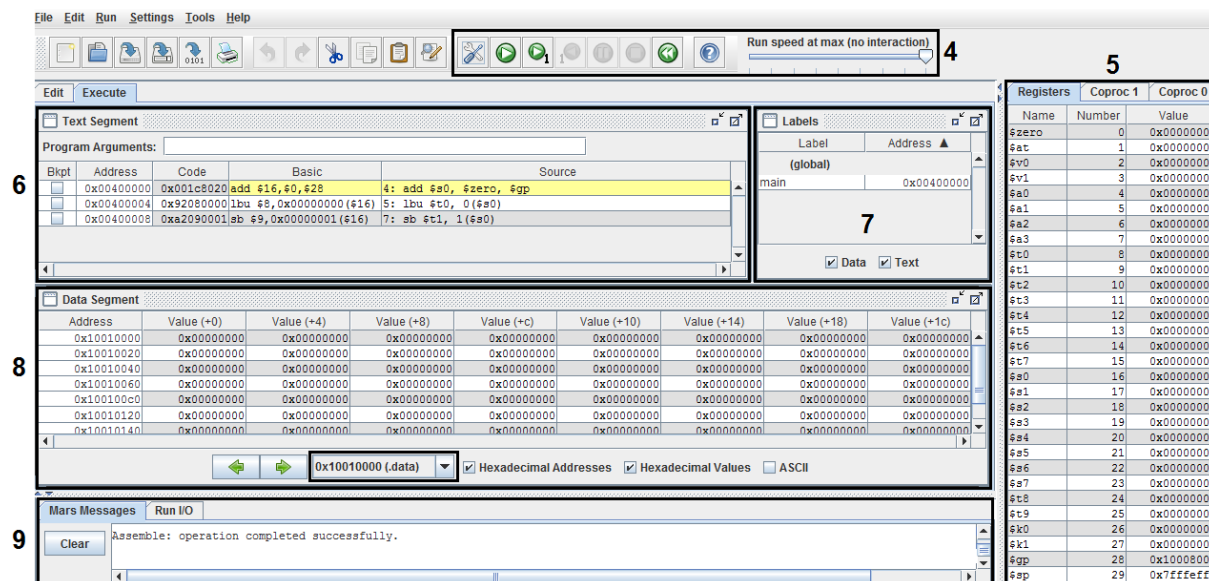


Fig. 2 – Janela de Execução do MARS.

4 – Atalhos: principais menus e atalhos que possibilitam a montagem das instruções simbólicas, a execução do código binário e a configuração da velocidade de execução.

5 – Registers: mostra os valores armazenados nos registradores de propósito geral do MIPS. A opção **Coproc 1** apresenta os registradores que armazenam valores de variáveis representadas em ponto flutuante; a opção **Coproc 0** refere-se aos registradores de controle usados pelo núcleo do sistema operacional para tratamento de exceções e interrupções. O conteúdo dos registradores pode ser apresentado em decimal ou hexadecimal.

6 – Text Segment: corresponde à área de memória onde é alocado o código do programa. Nessa janela são visualizados os endereços de memória, o código de máquina das instruções (em hexadecimal) e os correspondentes mnemônicos e seus operandos em linguagem de montagem. Abaixo do título **Bkp (break point)** pode-se selecionar a linha de código que se deseja inserir uma parada de execução.

7 – Labels: apresenta os endereços de memória que correspondem aos endereços simbólicos (labels) associados a dados e instruções. Pode ser ativado através da opção **Settings → Show Labels Window**.

8 – Data Segment: mostra os valores armazenados na memória. O conteúdo visualizado dependerá do contexto selecionado através do *combo-box* no rodapé da janela *Data Segment*. Por exemplo, a opção **.data** refere-se ao segmento de dados estáticos usados pelo programa aplicativo. Já a opção **sp** refere-se ao segmento de dados dinâmicos que armazena a estrutura de dados chamada de pilha (*stack*), que é crucial no suporte à chamada de procedimentos. A opção **.kdata** refere-se ao segmento de dados de uso reservado para o núcleo (*kernel*) do sistema operacional. O usuário comum desenvolve programas e aplicativos e, por isso, deve trabalhar na região **.data**.

9 – Message: contém as mensagens geradas pelo MARS para o usuário. Geralmente são apresentadas mensagens sobre o carregamento do programa ou sobre a sua execução e, se for o caso, os erros ocorridos.

Exercícios

1. Leitura/Escrita de um byte da memória (1 posição de memória)

a) Crie um arquivo novo no editor do MARS (**File → New**) com o seguinte código:

```
.text                # diretiva para definir o início do Text Segment
.globl main          # diretiva para definir o início do código do usuário
main:                # endereço simbólico (label) que define o início do código do usuário
    add $s0, $zero, $gp    # copia o valor do $gp no registrador $s0
    lbu $t0, 0($s0)        # lê o byte da posição de memória [$s0 + 0] e copia no byte menos significativo (mais à direita)
                                do registrador $t0
    sb $t1, 1($s0)         # copia o byte menos significativo (mais à direita) do reg. $t1 na posição de memória indicada
                                por [$s0 + 1]
```

Observações:

- O símbolo **#** indica que o conteúdo que segue na linha é um comentário (ignorado pelo montador)
- Os comentários servem para elucidar informações sobre a lógica do programa e são muito importantes na documentação do programa em linguagem de máquina, pois é pouco representativa do significado do programa.
- O registrador **\$gp (global pointer)** é utilizado para o acesso ao segmento de dados estáticos do programa aplicativo (normalmente inicializado pelo sistema operacional para apontar para o meio do segmento de dados estáticos).
- Os registradores de uso geral do MIPS são todos de 32 bits. Os bytes nesses registradores são organizados da seguinte forma:

byte + significativo		byte - significativo	
byte 3	byte 2	byte 1	byte 0

- Os valores numéricos escritos em linguagem de montagem são por *default* decimais, para valores em hexadecimal deve-se preceder o número com **0x** (por exemplo, **0x2ACF77BC**).
- Organize seus programas em pastas adequadas, como por exemplo, **Lab01**.

Data Segment									
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)	
0x10008000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x10008020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x10008040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x10008060	0x00000000	0x00000000	0x10000000 (.extern)	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x10008080	0x00000000	0x00000000	0x10010000 (.data)	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x100080a0	0x00000000	0x00000000	0x10040000 (.heap)	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x100080c0	0x00000000	0x00000000	current \$gp	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x100080e0	0x00000000	0x00000000	current \$sp	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x10008100	0x00000000	0x00000000	0x00400000 (.text)	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x10008120	0x00000000	0x00000000	0x90000000 (.kdata)	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x10008140	0x00000000	0x00000000	0xffff0000 (MMIO)	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	

←

→

current \$gp

Hexadecimal Addresses

Hexadecimal Values

ASCII

Fig. 5 – Seleção da sessão current \$gp.

Selecione o endereço que estava armazenado no registrador \$gp (0x1000 8000). Clique duas vezes para abrir um novo valor para essa posição de memória, adicione o valor 64 (0x40) e pressione Enter. Esse processo é semelhante à alteração de uma célula numa planilha do Excel.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10008000	64	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10008020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10008040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10008060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

current \$gp ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

Fig. 6 – Atribuição do valor 64 (0x40) ao endereço 0x1000 8000 que, neste caso, é o valor inicial do \$gp.

e) Da mesma forma, atribua o valor 55 (0x37) ao registrador \$t1 (\$9).

Registers		Coproc 1	Coproc 0
Name	Number	Value	
\$a2	6	0x00000000	
\$a3	7	0x00000000	
\$t0	8	0x00000000	
\$t1	9	55	
\$t2	10	0x00000000	
\$t3	11	0x00000000	

Fig. 7 – Atribuição do valor 55 (0x37) ao registrador \$t1.

f) Agora aperte a tecla F7 ou o atalho 1. Esse comando permite rodar o programa instrução por instrução. Então, passo a passo, veja a alteração do registrador \$s0, \$t0 e o valor da posição de memória correspondente no Data Segment. Para inicializar a aplicação utilize o atalho (F12), e para rodar de uma única vez, o atalho (F5).

1 - Qual o valor final armazenado no registrador \$t0?

\$t0 = 0x_____

2 - Qual o endereço do byte menos significativo (byte 0) da palavra armazenada em 0x10008000 e o valor final nele armazenado?

Endereço: 0x_____ Valor: 0x_____

3 - Qual o endereço do byte 1 da palavra armazenada em 0x10008000 e o valor final nele armazenado?

Endereço: 0x_____ Valor: 0x_____

2. Leitura/Escreva de múltiplos bytes na memória. O objetivo deste exercício é verificar como o MARS mostra a sequência de bytes na memória.

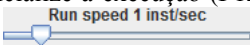
a) Crie o seguinte arquivo no editor do MARS:

```
.text
.globl main
main:
    add $s0, $zero, $gp    # copia o valor de $gp no registrador $s0
    lbu $t0, 0($s0)        # lê o byte da posição de memória [$s0+0] e copia o byte menos significativo de $t0
    lbu $t1, 1($s0)
    lbu $t2, 2($s0)
    lbu $t3, 3($s0)
    lbu $t4, 4($s0)
    lbu $t5, 5($s0)
    lbu $t6, 6($s0)
    lbu $t7, 7($s0)        # lê o byte da posição de memória [$s0+7] e copia o byte menos significativo de $t7
    lw  $t8, 0($s0)        # lê a word da posição de memória [$s0+0] e copia em $t8
```

b) Salve o seu programa (por exemplo, **Lab_01_Ex2.asm**).

c) Inicialize o valor 0x0FEE DDCC na posição de memória apontada por \$gp (0x1000 8000) e o valor 0x7788 99AA na posição apontada por \$gp+4 (0x1000 8004).

d) Execute o programa passo-a-passo. A cada passo, analise a modificação dos conteúdos dos registradores a partir do conteúdo da memória.

e) Re-inicialize a execução (F12), repita o passo c, configure a velocidade de execução para uma instrução por segundo  e rode a aplicação (F5).

Note que antes de executar o programa acima, você inicializou uma *word* (32 bits) no endereço apontado por \$gp e uma outra no endereço apontado por \$gp+4.

1 - Preencha a tabela abaixo com os valores observados ao final da execução do programa.

Endereço (0x)	Valor (0x)	Endereço (0x)	Valor (0x)
10008000		10008004	
10008001		10008005	
10008002		10008006	
10008003		10008007	

2 - Analise o conteúdo da tabela acima para escolher a afirmação correta. Para o sistema computacional simulado, dada uma palavra de memória, pode-se afirmar que:

- () os endereços de memória crescem do byte menos significativo para o byte mais significativo.
- () os endereços de memória crescem do byte mais significativo para o byte menos significativo.

3 - Analise o programa e, em particular, o efeito da instrução **lw \$t8, 0(\$s0)** para escolher a afirmação correta. Para o sistema computacional simulado, o endereço de uma palavra na memória corresponde:

- () ao mesmo endereço de memória de seu byte mais significativo (MSByte).
- () ao mesmo endereço de memória de seu byte menos significativo (LSByte).