

“Os humanos são pensadores lentos, desleixados e brilhantes. Computadores são rápidos, cuidadosos e estúpidos.” (John Pfeiffer)

Programas em Assembly, I/O e exercícios

Paulo Ricardo Lisboa de Almeida



Simulador MARS

- MARS (MIPS Assembler and Runtime Simulator)
 - Simulador MIPS
 - Desenvolvido por Pete Sanderson e Kenneth Vollmar
- Gratuito e de código aberto
 - Licença MIT
 - Download: <https://courses.missouristate.edu/KenVollmar/MARS/download.htm>
- O programa é um jar, então basta ter o Java instalado na sua máquina
- Maneira simples de executar em casa:
 - Abra um terminal no diretório onde o jar foi salvo
 - Digite (assumindo que o nome do jar baixado é Mars4_5.jar)
 - `java -jar Mars4_5.jar`
 - **Nas máquinas do Lab já existe um atalho pronto no “Menu Iniciar”**

Simulador MARS

/home/paulo/Área de Trabalho/programas/entradaSaida.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

Bkpt	Address	Code	Basic		
<input type="checkbox"/>	0x00400000	0x24020005	addiu \$2,\$0,0x00000005	4:	li \$v0, 5 #5 em \$v0 para S.O
<input type="checkbox"/>	0x00400004	0x0000000c	syscall	5:	syscall #chama o S.O. e re
<input type="checkbox"/>	0x00400008	0x20440014	addi \$4,\$2,0x00000014	6:	addi \$a0, \$v0, 20 #soma o resultado
<input type="checkbox"/>	0x0040000c	0x24020001	addiu \$2,\$0,0x00000001	7:	li \$v0, 1 #1 em \$v0 para S.O
<input type="checkbox"/>	0x00400010	0x0000000c	syscall	8:	syscall #chama o S.O. para
<input type="checkbox"/>	0x00400014	0x3404000a	ori \$4,\$0,0x0000000a	9:	ori \$a0,\$zero,10 #10 é o cc
<input type="checkbox"/>	0x00400018	0x2402000b	addiu \$2,\$0,0x0000000b	10:	li \$v0, 11 #11 em \$v0 para S.
<input type="checkbox"/>	0x0040001c	0x0000000c	syscall	11:	syscall #chama o S.O. para
<input type="checkbox"/>	0x00400020	0x34040039	ori \$4,\$0,0x00000039	12:	ori \$a0,\$zero, 57
<input type="checkbox"/>	0x00400024	0x24020001	addiu \$2,\$0,0x00000001	13:	li \$v0, 1
<input type="checkbox"/>	0x00400028	0x0000000c	syscall	14:	syscall
<input type="checkbox"/>	0x0040002c	0x2402000a	addiu \$2,\$0,0x0000000a	17:	li \$v0, 10 #10 em \$v0 para ir
<input type="checkbox"/>	0x00400030	0x0000000c	syscall	18:	syscall #chama o S.O. para

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$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	0x10008000
\$sp	29	0x7ffefffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Mars Messages Run I/O

Clear

Assemble: assembling /home/paulo/Área de Trabalho/programas/entradaSaida.asm

Assemble: operation completed successfully.

■ Simulador MARS

- Abra o MARS
- Crie um novo arquivo
- Salve como **aula.asm**

Programas em Assembly

- Programas escritos em assembly geralmente são salvos com a extensão **.s**, **.as** ou **.asm**
- Colocamos apenas um nível de indentação
 - Mais a esquerda
 - **Definições de rótulos e importações**
 - A uma tabulação de distância
 - **Todos os comandos**
- **Sempre comente o máximo possível em seus programas**

Criando seu programa

- Inicialmente, nossos programas terão o seguinte formato:

```
.text      ←————— Indica o início das instruções do programa
.globl main ←————— Indica que o rótulo main é visível globalmente
main:      ←————— Indica o ponto de início do programa
#seu programa aqui
end:       ←————— As instruções abaixo do end informam ao S.O. que o
li $v0, 10 programa terminou através de uma syscall.
syscall
```

Exemplo

- Escreva o seguinte no programa aula.asm

```
.text
.globl main
main:
    li $s5, 15    #pseudoinstrução para carregar o imediato 15 em $s5
end:
    li $v0, 10    #10 em $v0 para indicar ao S.O. que o programa acabou
    syscall      #chamar o S.O.
```

Assemble

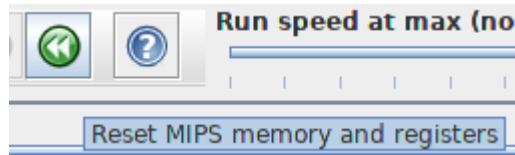
- Para montar o código e torná-lo executável, clique no botão “Assemble”



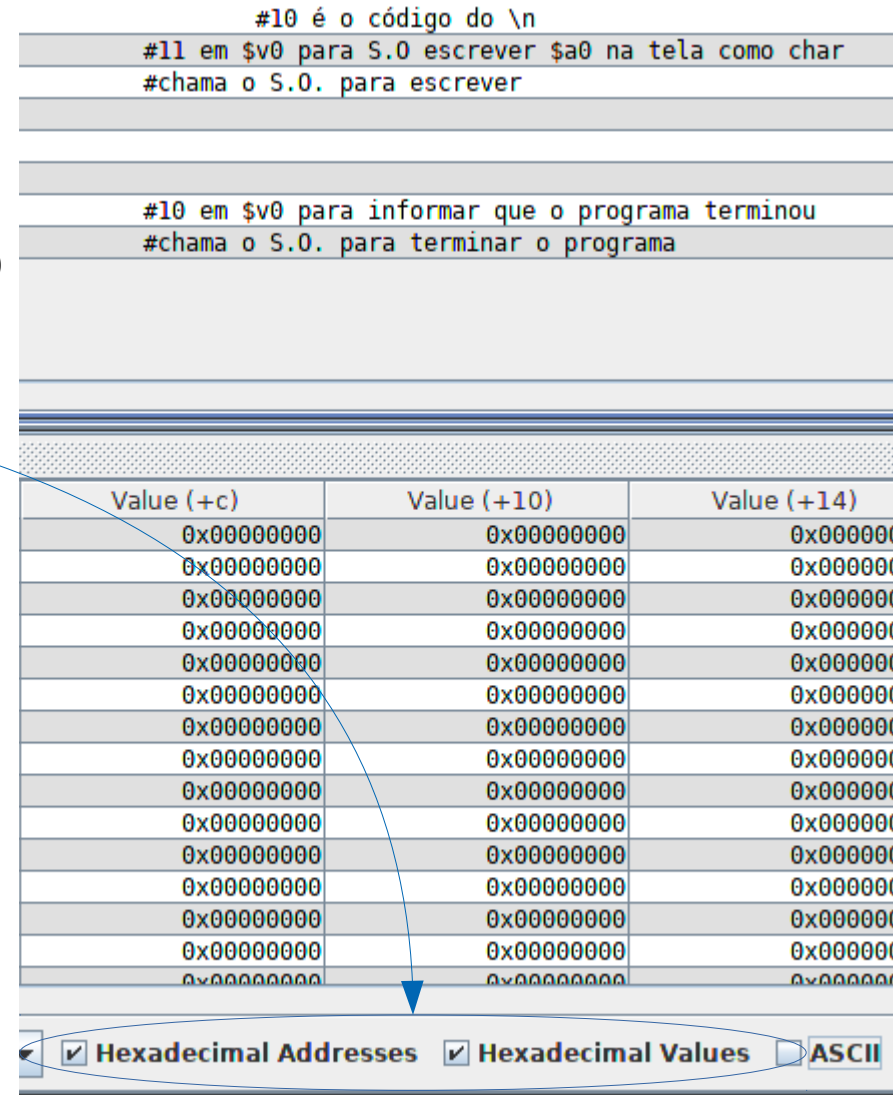
- Aperte o “play” e veja o resultado do seu programa nos registradores
- **Note que o li é uma pseudoinstrução, e foi traduzida para instruções reais**

Dicas

- Sempre verifique o formato em que os valores estão sendo exibidos nos registradores
 - Binário ou hexa
- Você pode zerar os registradores na interface
 - Necessário se deseja executar o programa novamente (para zerar o PC)



- O MARS segue uma implementação específica do MIPS, então alguns detalhes podem ser diferentes dos apresentados em Patterson e Hennessy (2019).



Exercícios

1. Faça um programa que calcule $9!$ e armazene o resultado em `$s0`
Veja como funciona a instrução `mul`. Exemplo:
`mul $s1,$s2,$s3`

Entrada e Saída

- A entrada e saída é feita com ajuda do Sistema Operacional
 - O MARS inclui um Sistema Operacional minimalista para simulações
- Precisamos fazer **syscalls**
 - Colocamos o código da operação desejada em **\$v0**
 - Instrução syscall devolve o controle ao S.O., que olha para \$v0 e faz o requisitado
 - Entradas e saídas são feitas na tela de RUN I/O
- Veja algumas chamadas de sistema válidas em
 - courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html

Exercícios

2. Modifique o exercício 1 para que seja calculado $n!$, onde n é solicitado do usuário. Exiba o resultado no console. Além disso, utilize a instrução `mult` no lugar de `mul`. Como `mult` funciona? Por que o resultado é armazenado em dois registradores?
3. Faça um programa que peça continuamente valores inteiros ao usuário. O programa termina quando o usuário digita -1. Ao final o programa deve exibir a soma e a média dos valores digitados.
4. Modifique o programa do exercício anterior, de forma que o programa termina quando o usuário digita -1, ou quando a soma atingir um valor maior ou igual a 2048.
5. Faça um programa que calcule o vigésimo número da sequência de Fibonacci e o armazene o registrador `$s1`.
6. Modifique o exercício anterior para que o índice do número de fibonacci a ser impresso seja requisitado do usuário, e o resultado seja impresso no console.

Exercícios

7. Crie um programa para um caixa eletrônico que calcula o menor número possível de cédulas que deve ser entregue a um usuário quando ele fizer um saque. Considere que a entrada do programa é o valor do saque, e a saída são as notas que o usuário receberá. Exiba as quantidades de notas como inteiros simples na tela, na seguinte ordem: notas de 50, 20, 10 e 5 reais, e moedas de 1 real. Exemplo se o usuário solicitar um saque de 87 reais:

1 1 1 1 2

Dica. Para imprimir um espaço :

li \$a0, 32 #32 é o código do espaço

li \$v0, 11 #11 em \$v0 para S.O escrever \$a0 na tela como char

syscall #chama o S.O. para escrever

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores: Interface Hardware/Software**. 5a Edição. Elsevier Brasil, 2019.
- Andrew S. Tanenbaum. **Organização estruturada de computadores**. 5. ed. São Paulo: Pearson, 2007.
- Harris, D. and Harris, S. **Digital Design and Computer Architecture**. 2a ed. 2012.
- courses.missouristate.edu/KenVollmar/mars/