

7 Programação em *Assembly* do MIPS

Objetivos: São dois os objetivos deste laboratório: (i) aprender usar o simulador MARS; e (ii) escrever e testar um programa completo em *assembly*.

Preparação: Veja MARS_Tutorial.pdf e MARS_features.pdf em <http://www.inf.ufpr.br/roberto/ci210/assembly>.

7.1 Programa que Computa a Série de Fibonacci

Para executar o simulador MARS diga:

```
java -jar /home/soft/linux/mips/Mars.jar
```

Possivelmente, seria uma boa ideia acrescentar ao seu ~/.bashrc a função que contenha esta linha de comando. Edite ~/.bashrc e acrescente o seguinte, preferencialmente no final do arquivo:

```
function mars() { java -jar /home/soft/linux/mips/Mars.jar "$@" ; }
```

Isso feito, diga `source ~/.bashrc` e então invoque o simulador dizendo apenas `mars`.

Copie <http://www.inf.ufpr.br/roberto/ci210/assembly/fibonacci.s> para a sua área de trabalho e siga as instruções em MARS_Tutorial.pdf.

7.2 Programa que Computa o Fatorial

Copie <http://www.inf.ufpr.br/roberto/ci210/assembly/fatorial.s> para a sua área de trabalho e verifique se o programa produz resultados corretos. Se encontrar algum erro, **corrija-o** e verifique sua solução.

wget <http://www.inf.ufpr.br/roberto/ci210/assembly/fatorial.s>

Programa 1: fatorial.s

```

1  # void main(void) { // fatorial iterativo
2
3  #      int i,j;
4      .data                # início da seção de dados
5  vi:      .word 1          # aloca espaço para vars globais
6  vj:      .word 1          # estas duas são apenas para exemplificar
7
8  resp:    .asciiz "fatorial de 5 = "      # string com '\0'
9
10
11      .text                # início da seção de código
12      .globl main          # define main como nome global
13  main:
14
15      #      j=1;
16      addi $t1, $zero, 1      # este é o endereço de main()
17
18      #      for(i=1; i <= n; i++)
19      addi $t4, $zero, 1
20
21  for:     slti $t7, $t4, 6      # n < 6      fat(5)
22      beq $t7, $zero, fimfor    # t7 == false -> fimfor
23
24      #      j = j*i;
25      mult $t1, $t4            # resultado em 64 bits (HI & LO)
26      mflo $t4                # seleciona 32 bits menos sign
27
28      addi $t4, $t4, 1
29
30      j for
31
32  fimfor:  addi $v0, $zero, 4      # imprime resposta
33      la $a0, resp              # syscall(4) = imprime string
34      syscall
35
36      addi $v0, $zero, 1          # imprime inteiro
37      addu $a0, $zero, $t1        # syscall(1) = imprime inteiro
38      syscall
39
40      #      return(0);
41      li $v0, 10                # termina programa
42      syscall                  # syscall(10) = termina programa
43
44      # }

```

Espaço em branco proposital.

7.3 Programa para Copiar *Strings*

Traduza o Programa 2 para *assembly* do MIPS e verifique sua corretude com MARS. Use o código que imprime inteiros em `fibonacci.s` como modelo para o `printf()`, alterando o tipo de saída de inteiro para *string*. Os códigos das *syscalls* estão definidos em

Help → MIPS → Syscalls.

A instrução **lb** `rt, desl(rs)` (*load-byte*) carrega o byte apontado por `(extSinal(desl)+rs)` no registrador `rt`, estendendo o sinal (b_7).

A instrução **lbu** `rt, desl(rs)` (*load-byte unsigned*) é similar à **lb** mas não estende o sinal do byte carregado; os três bytes mais significativos são preenchidos com zero.

A instrução **sb** `rt, desl(rs)` (*store-byte*) armazena o byte menos significativo em `rt` no endereço apontado por `(extSinal(desl)+rs)`.

Para alocar as *strings* em memória use a linha 8 do Programa 1 como exemplo. A diretiva `.asciiz` aloca uma *string* incluindo o `'\0'`, enquanto que `.ascii` aloca uma *string* sem o `'\0'`. Veja o tutorial do Mars para a lista das diretivas que este provê, ou

Help → MIPS → Directives.

Repare que as diretivas aceitas pelo montador do Mars são diferentes daquelas aceitas por `mips-as`.

Programa 2: `strcpy.c`

```

1  char fte[16]="abcd-efgh-ijkl-";
2  char dst[16]= { '\0' };           // inicializa com 16 '\0'
3
4  void main (void) {
5      int i,f,n;
6
7      i = 0;                        // inclui '\0' na contagem
8      // copia e computa tamanho da cadeia, inclusive '\0'
9      while( (dst[i] = fte[i]) != '\0' ) // atribui e então compara
10         i = i + 1;
11
12     i = i + 1;                     // inclui '\0' na contagem
13
14     // sua versão assembly de printf() deve ter um
15     // número fixo de argumentos e usar as syscalls do MARS
16     printf("fonte:_%s\n", fte);
17     printf("dest:_%s\n", dst);
18     printf("tam:_%d\n", i);
19 }
```

Num programa em C, as variáveis são declaradas, tipicamente no início do programa, e então escrevemos o código das funções. Em *assembly*, o espaço em memória para as variáveis é alocado na seção `.data`, através de diretivas tais como `.space`, `.byte` e `.word`. Por exemplo, um vetor para 32 caracteres, chamado de `buf`, é declarado com

```

.data          # seção com dados
buf: .size 32   # char buf[32];
```

Seu programa em *assembly* pode referenciar este vetor com

```

.text          # seção com código
...
la    t0, buf    # t0 = &(buf[0]);
lbu   t5, 0(t0)   # t5 = *t0;
addi  t0, t0, 1   # t0++;
```

7.4 Mais do Fatorial

Este exercício é para aqueles que já utilizaram o Mars em outras disciplinas, e que chegaram a esta tarefa com tempo disponível. Para aqueles que utilizam Mars pela primeira vez, este exercício deve ser tentado fora do horário de aula.

Traduza o Programa 3 para *assembly* do MIPS e verifique sua corretude com MARS.

Programa 3: Duas versões do fatorial iterativo

```

1  void main (void) {
2      int i,f,n;
3
4      n = 5;
5      i = f = 1;
6
7      do {
8          f = f * i;
9          i = i + 1;
10     } while (i <= n);
11
12     printf("%d□%d\n",n,f); // use syscalls do MARS
13
14     f = 1;
15     i = n;
16     while (i > 0) {
17         f = f * i;
18         i = i - 1;
19     }
20     printf("%d□%d\n",n,f); // use syscalls do MARS
21
22     return(0);
23 }
```

Espaço em branco proposital.

Referências

[RH17] *CI064 – Software Básico*, Roberto A Hexsel, 2017, <http://www.inf.ufpr.br/roberto/ci064/swbas.pdf>

Histórico das Revisões:

12nov19: ajustes cosméticos;
01nov18: ajustes cosméticos;
09nov17: correção em strcpy, defs de lb, lbu, .data .text;
15jan16: programas com números de linha;
17mar14: terceira versão, adição para alunos que já usaram MARS;
23out13: segunda versão, sem funções;
08jun13: primeira versão.