

Construção de um compilador de Lua para Parrot Virtual Machine usando Objective Caml

Guilherme Pacheco de Oliveira

`guilherme.061@gmail.com`

Faculdade de Computação
Universidade Federal de Uberlândia

16 de agosto de 2016

Lista de Figuras

2.1	Instalando e testando LUA	8
2.2	Instalando e testando OCaml	9
2.3	Instalando e testando Parrot	10

Lista de Tabelas

Lista de Listagens

2.1	Output Simples em Parrot Assembly Language	10
2.2	Output Simples em Parrot Intermediate Representation	10
3.1	Programa nano 01 em Lua	12
3.2	Programa nano 01 em PASM	12
3.3	Programa nano 02 em Lua	12
3.4	Programa nano 02 em PASM	12
3.5	Programa nano 03 em Lua	13
3.6	Programa nano 03 em PASM	13
3.7	Programa nano 04 em Lua	13
3.8	Programa nano 04 em PASM	13
3.9	Programa nano 05 em Lua	13
3.10	Programa nano 05 em PASM	13
3.11	Programa nano 06 em Lua	13
3.12	Programa nano 06 em PASM	14
3.13	Programa nano 07 em Lua	14
3.14	Programa nano 07 em PASM	14
3.15	Programa nano 08 em Lua	14
3.16	Programa nano 08 em PASM	14
3.17	Programa nano 09 em Lua	15
3.18	Programa nano 09 em PASM	15
3.19	Programa nano 10 em Lua	15
3.20	Programa nano 10 em PASM	16
3.21	Programa nano 11 em Lua	16
3.22	Programa nano 11 em PASM	16
3.23	Programa nano 12 em Lua	17
3.24	Programa nano 12 em PASM	17
3.25	Programa micro 01 em Lua	17
3.26	Programa Micro 01 em PASM	18
3.27	Programa micro 02 em Lua	18
3.28	Programa Micro 02 em PASM	19
3.29	Programa micro 03 em Lua	19
3.30	Programa Micro 03 em PASM	19
3.31	Programa micro 04 em Lua	20
3.32	Programa Micro 04 em PASM	20
3.33	Programa micro 05 em Lua	21
3.34	Programa Micro 05 em PASM	22
3.35	Programa micro 06 em Lua	23
3.36	Programa Micro 06 em PASM	23
3.37	Programa micro 07 em Lua	24
3.38	Programa Micro 07 em PASM	24

3.39 Programa micro 08 em Lua	25
3.40 Programa micro 09 em Lua	25
3.41 Programa micro 10 em Lua	25
3.42 Programa micro 11 em Lua	26

Sumário

Lista de Figuras	2
Lista de Tabelas	3
1 Introdução	7
2 Instalação dos componentes	8
2.1 Homebrew	8
2.2 Lua	8
2.2.1 Instalação e Teste	8
2.2.2 Informações sobre a linguagem Lua	9
2.3 Ocaml	9
2.3.1 Instalação e Teste	9
2.3.2 Informações sobre a linguagem OCaml	9
2.4 Parrot Virtual Machine	9
2.4.1 Instalação e Teste	9
2.4.2 Informações sobre a Parrot Virtual Machine	10
2.4.3 Parrot Assembly Language (PASM)	11
2.4.4 Parrot Intermediate Representation (PIR)	11
3 Códigos LUA e Parrot Assembly (PASM)	12
3.0.1 Nano Programas	12
3.0.2 Micro Programas	17
4 Referências	27

Capítulo 1

Introdução

Este documento foi escrito para documentar o processo de instalação de todas as ferramentas necessárias para a construção de um compilador da Linguagem Lua para a máquina virtual Parrot, utilizando a linguagem Ocaml para fazer a implementação.

Um segundo objetivo é mostrar uma série de programas simples na linguagem Lua e sua versão na linguagem PASM, que é a linguagem assembly utilizada pela Parrot, afim de estabelecer um guia sobre a saída dos programas que passarão pelo compilador.

Outro objetivo é adquirir conhecimento sobre a linguagem Lua, ter um contato inicial com OCaml e conhecer como funciona a máquina virtual Parrot, suas linguagens de Assembly e bytecode e de compiladores já existentes

O Sistema Operacional utilizado é OS X El Capitan 10.11.6

Capítulo 2

Instalação dos componentes

2.1 Homebrew

Homebrew é um gerenciador de pacotes para Mac OS X, escrito em Ruby, e é responsável por instalar pacotes nos diretórios adequados e fazer adequadamente a configuração desses pacotes, instalá-lo facilita todo o processo de instalação dos componentes necessários.

Para instalar o homebrew basta digitar no terminal:

```
\$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2.2 Lua

2.2.1 Instalação e Teste

Para instalar Lua através do homebrew, basta digitar no terminal:

```
\$ brew install lua
```

Resultado:

Figura 2.1: *Instalando e testando LUA*

```
oliveira@lua oliveira$ brew install lua
=> Downloading https://homebrew.bintray.com/bottles/lua-5.2.4.3.el_capitan.bott
##### 100.0%
=> Pouring Lua-5.2.4.3.el_capitan.bottle.tar.gz
=> Caveats
Please be aware due to the way Luarocks is designed any binaries installed
via Luarocks-5.2 AND 5.1 will overwrite each other in /usr/local/bin.

This is, for now, unavoidable. If this is troublesome for you, you can build
rocks with the --tree= command to a special, non-conflicting location and
then add that to your $PATH.
=> Summary
[0] /usr/local/Cellar/lua/5.2.4.3: 143 files, 697.3K
oliveira@lua oliveira$ lua
Lua 5.2.4 Copyright (C) 1994-2015 Lua.org, PUC-Rio
> print("Hello World")
Hello World
> ^D
oliveira@lua oliveira$
```


2.2.2 Informações sobre a linguagem Lua

A principal referência para Lua é a documentação em seu site oficial [1]. Lua é uma linguagem de programação de extensão, projetada para dar suporte à outras linguagens de programação procedimental e planejada para ser usada como uma linguagem de script leve e facilmente embarcável, é implementada em C.

2.3 Ocaml

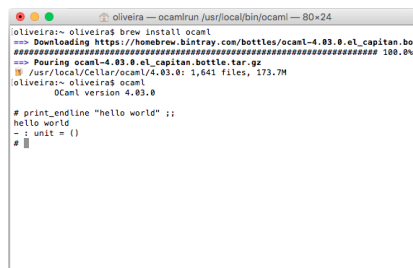
2.3.1 Instalação e Teste

Novamente através do homebrew, basta digitar:

```
\$ brew install ocaml
```

Resultado:

Figura 2.2: *Instalando e testando OCaml*



```
oliveira~ oliveira$ brew install ocaml
=> Downloading https://homebrew.bintray.com/bottles/ocaml-4.03.0.el_capitan.bot
=> Pouring ocaml-4.03.0.el_capitan.bottle.tar.gz
=> /usr/local/Cellar/ocaml/4.03.0: 1,641 files, 173.7M
oliveira~ oliveira$ ocaml
OCaml version 4.03.0

# print_endline "hello world" ;;
hello world
- : unit = ()
#
```

2.3.2 Informações sobre a linguagem OCaml

A documentação oficial do OCaml [2] possui manuais, licenças, documentos e algumas dicas sobre como programar adequadamente na linguagem. OCaml é uma linguagem de programação funcional, imperativa e orientada à objetos.

2.4 Parrot Virtual Machine

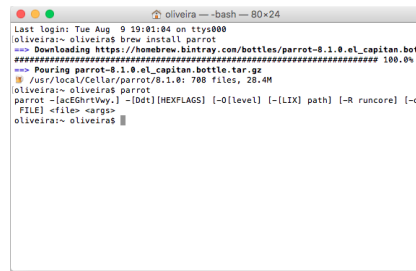
2.4.1 Instalação e Teste

Digitar no Terminal:

```
\$ brew install parrot
```

Resultado:

Figura 2.3: Instalando e testando Parrot



```

oliveira ~ - bash -- 80x24
Last login: Tue Aug 9 19:01:04 on ttys000
oliveira:~ oliveira$ brew install parrot
=> Downloading https://homebrew.bintray.com/bottles/parrot-8.1.0.el_capitan.bot
##### 100.0%
=> Pouring parrot-8.1.0.el_capitan.bottle.tar.gz
  > /usr/local/Cellar/parrot/8.1.0: 706 files, 28.4M
oliveira:~ oliveira$ parrot
parrot -[acGhrtWxy.] -[Ddt][HEXFLAGS] [-O[level]] [-[LIX] path] [-R runcore] [-o FILE] <file> <args>
oliveira:~ oliveira$

```

2.4.2 Informações sobre a Parrot Virtual Machine

A máquina virtual Parrot é utilizada principalmente para linguagens dinâmicas como Perl, Python, Ruby e PHP, seu design foi originalmente feito para trabalhar com a versão 6 de Perl, mas seu uso foi expandido como uma máquina virtual dinâmica e de propósito geral, apta a lidar com qualquer linguagem de programação de alto nível. [3]

Parrot pode ser programada em diversas linguagens, os dois mais utilizados são: Parrot Assembly Language (PASM): É a linguagem de mais baixo nível utilizada pela Parrot, muito similar a um assembly tradicional. Parrot Intermediate Representation(PIR): De mais alto nível que PASM, também um pouco mais facil de se utilizar e mais utilizada.

Fazendo alguns testes com PASM e PIR:

Listagem 2.1: Output Simples em Parrot Assembly Language

```

1 say "Here are the news about Parrots."
2 end

```

Para executar o código:

```
\$ parrot news.pasm
```

Listagem 2.2: Output Simples em Parrot Intermediate Representation

```

1 .sub main :main
2   print "No parrots were involved in an accident on the M1 today...\n"
3 .end

```

Para executar o código:

```
\$ parrot hello.pir
```

Os arquivos PASM e PIR são convertidos para Parrot Bytecode (PBC) e somente então são executados pela máquina virtual, é possível obter o arquivo .pbc através comando:

```
\$ parrot -o output.pbc input.pasm
```

Apesar da documentação oficial enfatizar que PIR é mais utilizado e mais recomendado para o desenvolvimento de compiladores para Parrot, o alvo será a linguagem Assembly PASM.

2.4.3 Parrot Assembly Language (PASM)

A linguagem PASM é muito similar a um assembly tradicional, com exceção do fato de que algumas instruções permitem o acesso a algumas funções dinâmicas de alto nível do sistema Parrot.

Parrot é uma máquina virtual baseada em registradores, há um número ilimitado de registradores que não precisam ser instanciados antes de serem utilizados, a máquina virtual se certifica de criar os registradores de acordo com a sua necessidade, tal como fazer a reutilização e se livrar de registradores que não estão mais sendo utilizados, todos os registradores começam com o símbolo "\$" e existem 4 tipos de dados, cada um com suas regras:

Strings: Registradores de strings começam com um S, por exemplo: "\$S10"

Inteiros: Registradores de inteiros começam com um I, por exemplo: "\$I10"

Número: Registradores de números de ponto flutuante, começam com a letra N, por exemplo: "\$N10"

PMC: São tipos de dados utilizados em orientação a objetos, podem ser utilizados para guardar vários tipos de dados, começam com a letra P, por exemplo: "\$P10"

Para mais referências sobre PASM, consultar [4], [7] para os opcodes e [8] para exemplos.

2.4.4 Parrot Intermediate Representation (PIR)

A maioria dos compiladores possuem como alvo o PIR, inclusive o que será utilizado para estudar qual o comportamento um compilador deve ter ao gerar o assembly. A própria máquina virtual Parrot possui um módulo intermediário capaz de interpretar a linguagem PIR e gerar o bytecode ou o próprio assembly (PASM), além disso, existem compiladores capazes de realizar a mesma tarefa.

PIR é de nível mais alto que assembly mas ainda muito próximo do nível de máquina, o principal benefício é a facilidade em programar em PIR em comparação com a programação em PASM, além disso, ela foi feita para compiladores de linguagens de alto nível gerarem código PIR para trabalhar com a máquina Parrot. Mais informações sobre PIR e sua sintaxe podem ser encontradas em [5].

Capítulo 3

Códigos LUA e Parrot Assembly (PASM)

3.0.1 Nano Programas

Nano 01

Listagem 3.1: Programa nano 01 em Lua

```
1 -- Listagem 1: Mo dulo mi nimo que caracteriza um programa
```

Listagem 3.2: Programa nano 01 em PASM

```
1 # Modulo Minimo
2 end
```

Nano 02

Listagem 3.3: Programa nano 02 em Lua

```
1 -- Listagem 2: Declarac a o de uma varia vel
2
3 -- Em Lua, declaração de variaveis limitam apenas seu escopo
4 -- As variaveis podem ser local ou global
5 -- local: local x = 10 - precisam ser inicializadas
6 -- global: x = 10      - não precisam ser inicializadas
7 -- local x             é um programa aceito em lua (declaração de uma
    variavel local)
8 -- x                   não é um programa aceito em lua
```

Listagem 3.4: Programa nano 02 em PASM

```
1 # Declarando uma variavel
2
3 end
```

Nano 03

Listagem 3.5: Programa nano 03 em Lua

```
1 -- Atribuicao de um inteiro a uma variavel
2 n = 1
```

Listagem 3.6: Programa nano 03 em PASM

```
1 # Atribuição de um inteiro a uma variavel
2
3 set I1, 1
4 end
```

Nano 04

Listagem 3.7: Programa nano 04 em Lua

```
1 -- Atribuição de uma soma de inteiros a uma variavel
2 n = 1 + 2
```

Listagem 3.8: Programa nano 04 em PASM

```
1 # Atribuição de uma soma de inteiros a uma variavel
2 set I1, 1
3 set I2, 2
4 add I3, I1, I2
5 end
```

Nano 05

Listagem 3.9: Programa nano 05 em Lua

```
1 -- Inclusão do comando de impressão
2 n = 2
3 print(n)
```

Listagem 3.10: Programa nano 05 em PASM

```
1 # Inclusão do comando de impressão
2 set I1, 2
3 print I1
4 print "\n"
5
6 end
```

Nano 06

Listagem 3.11: Programa nano 06 em Lua

```
1 -- Listagem 6: Atribuição de uma subtração de inteiros a uma
   variavel
2
3 n = 1 - 2
4 print(n)
```

Listagem 3.12: Programa nano 06 em PASM

```

1 # Atribuição de uma subtração de inteiros a uma variável
2 set I1, 1
3 set I2, 2
4 sub I3, I1, I2
5
6 print I3
7 print "\n"
8
9 end

```

Nano 07

Listagem 3.13: Programa nano 07 em Lua

```

1 -- Listagem 7: Inclusa o do comando condicional
2 n = 1
3 if (n == 1)
4 then
5     print(n)
6 end

```

Listagem 3.14: Programa nano 07 em PASM

```

1 # Inclusão do comando condicional
2
3 set     I1, 1 # atribuição
4 eq     I1, 1, VERDADEIRO
5 branch FIM
6
7 VERDADEIRO:
8 print  I1
9 print  "\n"
10
11 FIM:
12 end

```

Nano 08

Listagem 3.15: Programa nano 08 em Lua

```

1 -- Listagem 8: Inclusa o do comando condicional com parte sena o
2
3 n = 1
4 if(n == 1)
5 then
6     print(n)
7 else
8     print("0")
9 end

```

Listagem 3.16: Programa nano 08 em PASM

```

1 # Inclusão do comando condicional senão
2
3 set     I1, 1

```

3.0

```
4 eq      I1, 1, VERDADEIRO
5 print   "0\n"
6 branch  FIM
7
8 VERDADEIRO:
9 print   I1
10 print  "\n"
11
12 FIM:
13 end
```

Nano 09

Listagem 3.17: Programa nano 09 em Lua

```
1 -- Listagem 9: Atribuição de duas operações aritméticas sobre
   inteiros a uma variável
2
3 n = 1 + 1 / 2
4 if (n == 1)
5 then
6     print(n)
7 else
8     print("0")
9 end
```

Listagem 3.18: Programa nano 09 em PASM

```
1 # Atribuição de duas operações aritméticas sobre inteiros a uma variável
2
3 set      I1, 1
4 set      I2, 2
5 div      I3, I1, I2
6 add      I4, I1, I3
7
8 eq       I4, 1, VERDADEIRO
9 print    "0\n"
10 branch  FIM
11
12 VERDADEIRO:
13 print    I4
14 print    "\n"
15
16 FIM:
17 end
```

Nano 10

Listagem 3.19: Programa nano 10 em Lua

```
1 -- Listagem 10: Atribuição de duas variáveis inteiras
2 n = 1
3 m = 2
4
5 if (n == m)
6 then
7     print(n)
```

```

8 else
9   print("0")
10 end

```

Listagem 3.20: Programa nano 10 em PASM

```

1 # Atribuição de duas variáveis inteiras
2
3 set      I1, 1
4 set      I2, 2
5
6 eq I1, I2, VERDADEIRO
7 print    "0\n"
8 branch   FIM
9
10 VERDADEIRO:
11 print    I1
12 print    "\n"
13
14 FIM:
15 end

```

Nano 11

Listagem 3.21: Programa nano 11 em Lua

```

1 -- Listagem 11: Introduc a o do comando de repetiç a o enquanto
2 n = 1
3 m = 2
4 x = 5
5
6 while(x > n)
7 do
8   n = n + m
9   print(n)
10 end

```

Listagem 3.22: Programa nano 11 em PASM

```

1 # Introdução do comando de repetição enquanto
2
3 set      I1, 1 # n
4 set      I2, 2 # m
5 set      I3, 5 # x
6
7 TESTE:
8 gt       I3, I1, LOOP # gt = greater then
9 branch   FIM
10
11 LOOP:
12 add      I1, I1, I2
13 print    I1
14 print    "\n"
15 branch   TESTE
16
17 FIM:
18 end

```

Nano 12

Listagem 3.23: Programa nano 12 em Lua

```

1 -- Listagem 12: Comando condicional aninhado em um comando de
    repetição
2 n = 1
3 m = 2
4 x = 5
5
6 while (x > n)
7 do
8     if (n == m)
9     then
10         print(n)
11     else
12         print("0")
13     end
14     x = x - 1
15 end

```

Listagem 3.24: Programa nano 12 em PASM

```

1 # Comando condicional aninhado com um de repetição
2
3 set     I1, 1
4 set     I2, 2
5 set     I3, 5
6
7 TESTE_ENQUANTO:
8 gt      I3, I1, LOOP
9 branch  FIM
10
11 LOOP:
12 eq      I1, I2, VERDADEIRO
13 print   "0\n"
14 branch  POS_CONDICIONAL
15
16 VERDADEIRO:
17 print   I1
18 print   "\n"
19
20 POS_CONDICIONAL:
21 dec     I3          # decrementa I3 (x)
22 branch  TESTE_ENQUANTO
23
24 FIM:
25 end

```

3.0.2 Micro Programas

Micro 01

Listagem 3.25: Programa micro 01 em Lua

```

1 -- Listagem 13: Converte graus Celsius para Fahrenheit
2

```

```

3 -- [[ Funcao: Ler uma temperatura em graus Celsius e apresenta-la
    convertida em graus Fahrenheit. A formula de conversao é : F=(9*C
    +160) / 5, sendo F a temperatura em Fahrenheit e C a temperatura em
    Celsius. --]]
4
5 print("Tabela de Conversão: Celsius -> Fahrenheit")
6 print("Digite a Temperatura em Celsius: ")
7 cel = io.read("*number")
8 far = (9*cel+160)/5
9 print("A nova temperatura é:", far)

```

Listagem 3.26: Programa Micro 01 em PASM

```

1 # Converte graus Celsius para Fahrenheit
2 .loadlib 'io_ops'           # Para fazer IO
3
4 set      S1, "Tabela de Conversao: Celsius -> Fahrenheit\n"
5 set      S2, "Digite a Temperatura em Celsius: "
6 set      S3, "A nova temperatura e: "
7 set      S4, " graus F."
8
9 print    S1
10 print   S2
11 read    S10, 5
12 set     I1, S10
13
14 mul     I1, I1, 9
15 add     I1, I1, 160
16 div     I1, I1, 5
17
18 print    S3
19 print    I1
20 print    S4
21 print    "\n"
22
23 end

```

Micro 02

Listagem 3.27: Programa micro 02 em Lua

```

1 -- Listagem 14: Ler dois inteiros e decide qual é maior
2
3 --[[ Funcao : Escrever um algoritmo que leia dois valores inteiro
    distintos e informe qual e o maior --]]
4
5 print("Escreva o primeiro número:")
6 num1 = io.read("*number")
7 print("Escreva o segundo número:")
8 num2 = io.read("*number")
9
10 if(num1 > num2)
11 then
12     print("O primeiro número", num1, "é maior que o segundo", num2)
13 else
14     print("O segundo número", num2, "é maior que o primeiro", num1)
15 end

```

Listagem 3.28: Programa Micro 02 em PASM

```

1 # Ler dois inteiros e decidir qual e maior
2 .loadlib 'io_ops'
3
4 set      S1, "Digite o primeiro numero: "
5 set      S2, "Digite o segundo numero: "
6 set      S3, "o primeiro numero"
7 set      S4, "o segundo numero"
8 set      S5, " e maior que "
9
10 print    S1
11 read     S10, 3
12 set      I1, S10
13 print    S2
14 read     S11, 3
15 set      I2, S11
16
17 gt       I1, I2, VERDADEIRO
18 print    S4
19 print    S5
20 print    S3
21 print    "\n"
22 branch   FIM
23
24 VERDADEIRO:
25 print    S3
26 print    S5
27 print    S4
28 print    "\n"
29
30 FIM:
31 end

```

Micro 03

Listagem 3.29: Programa micro 03 em Lua

```

1 -- Lê um número e verifica se ele está entre 100 e 200
2 --[[ Função: Faça um algoritmo que receba um número e diga se este número
   está no intervalo entre 100 e 200 --]]
3
4 print("Digite um número:")
5 numero = io.read("*number")
6
7 if(numero >= 100)
8 then
9     if(numero <= 200)
10 then
11     print("O número está no intervalo entre 100 e 200")
12 else
13     print("O número não está no intervalo entre 100 e 200")
14 end
15 else
16     print("O número não está no intervalo entre 100 e 200")
17 end

```

Listagem 3.30: Programa Micro 03 em PASM

```

1 # Le um numero e verifica se ele esta entre 100 e 200
2 .loadlib 'io_ops'
3
4 set      S1, "Digite um numero: "
5 set      S2, "O numero esta no intervalo entre 100 e 200\n"
6 set      S3, "O numero nao esta no intervalo entre 100 e 200\n"
7
8 print     S1
9 read      S10, 3
10 set      I1, S10
11
12 ge        I1, 100, MAIOR_QUE_100
13 branch    NAO_ESTA_NO_INTERVALO
14
15 MAIOR_QUE_100:
16 le        I1, 200, MENOR_QUE_200
17
18 NAO_ESTA_NO_INTERVALO:
19 print     S3
20 branch    FIM
21
22 MENOR_QUE_200:
23 print     S2
24
25 FIM:
26 end

```

Micro 04

Listagem 3.31: Programa micro 04 em Lua

```

1 -- Listagem 16: Lê números e informa quais estão entre 10 e 150
2
3 --[[ Função: Ler 5 números e ao final informar quantos números estão no
   intervalo entre 10 (inclusive) e 150(inclusive) --]]
4
5 intervalo = 0
6
7 for x=1,5,1
8 do
9   print("Digite um número")
10  num = io.read("*number")
11  if(num >= 10)
12  then
13    if(num <= 150)
14    then
15      intervalo = intervalo + 1
16    end
17  end
18 end
19
20 print("Ao total, foram digitados",intervalo,"números no intervalo entre 10
   e 150")

```

Listagem 3.32: Programa Micro 04 em PASM

```

1 # Le numeros e informa quais estao entre 10 e 150
2 .loadlib 'io_ops'

```

3.0

```
3
4 set      S1, "Digite um numero: "
5 set      S2, "Ao total foram digitados "
6 set      S3, " numeros no intervalo entre 10 e 150."
7
8 set      I1, 1                                # x
9 set      I2, 0                                # intervalo
10
11 LOOP_TESTE:
12 le       I1, 5, INICIO_LOOP
13 branch   FIM
14
15 INICIO_LOOP:
16 print    S1
17 read     S10, 3
18 set      I10, S10
19
20 ge       I10, 10, MAIOR_QUE_10
21 branch   FIM_LOOP
22
23 MAIOR_QUE_10:
24 le       I10, 150, MENOR_QUE_150
25 branch   FIM_LOOP
26
27 MENOR_QUE_150:
28 inc      I2
29
30 FIM_LOOP:
31 inc      I1
32 branch   LOOP_TESTE
33
34
35 FIM:
36 print    S2
37 print    I2
38 print    S3
39 print    "\n"
40 end
```

Micro 05

Listagem 3.33: Programa micro 05 em Lua

```
1 -- Listagem 17: Lê strings e caracteres
2 --[[ Função: Escrever um algoritmo que leia o nome e o sexo de 56 pessoas
   e informe o nome e se ela é homem ou mulher. No final informe o total
   de homens e mulheres --]]
3
4 h = 0
5 m = 0
6 for x=1,5,1
7 do
8   print("Digite o nome: ")
9   nome = io.read()
10  print("H - Homem ou M - Mulher")
11  sexo = io.read()
12  if(sexo == 'H') then h = h + 1
13  elseif (sexo == 'M') then m = m + 1
```

```

14  else print("Sexo só pode ser H ou M!")
15  end
16 end
17
18 print("Foram inseridos",h,"homens")
19 print("Foram inseridas",m,"mulheres")

```

Listagem 3.34: Programa Micro 05 em PASM

```

1 # Le strings e caracteres
2 .loadlib 'io_ops'
3
4 set      S2, "H - Homem ou M - Mulher: "
5 set      S3, "Sexo so pode ser H ou M!\n"
6 set      S4, "Foram inseridos "
7 set      S5, "Foram inseridas "
8 set      S6, " homens"
9 set      S7, " mulheres"
10
11 set      I1, 1          # x
12 set      I2, 0          # homens
13 set      I3, 0          # mulheres
14
15 LOOP_TESTE:
16 le       I1, 5, INICIO_LOOP
17 branch   FIM
18
19 INICIO_LOOP:
20 print    S2
21 read     S11, 2
22
23 eq       S11, "H\n", HOMEM
24 eq       S11, "M\n", MULHER
25
26 print    S3
27 branch   FIM_LOOP
28
29 HOMEM:
30 inc      I2
31 branch   FIM_LOOP
32
33 MULHER:
34 inc      I3
35
36 FIM_LOOP:
37 inc      I1
38 branch   LOOP_TESTE
39
40 FIM:
41 print    S4
42 print    I2
43 print    S6
44 print    "\n"
45
46 print    S5
47 print    I3
48 print    S7
49 print    "\n"
50 end

```

Micro 06

Listagem 3.35: Programa micro 06 em Lua

```

1 -- Escreve um número lido por extenso
2
3 --[[ Função: Faça um algoritmo que leia um número de 1 a 5 e o escreva por
    extenso. Caso o usuário digite um número que não esteja nesse
    intervalo, exibir mensagem: número invalido --]]
4
5 print("Digite um número de 1 a 5")
6 numero = io.read("*number")
7 if(numero == 1) then print("Um")
8 elseif (numero == 2) then print("Dois")
9 elseif (numero == 3) then print("Três")
10 elseif (numero == 4) then print("Quatro")
11 elseif (numero == 5) then print("Cinco")
12 else print("Número Invalido!!!")
13 end

```

Listagem 3.36: Programa Micro 06 em PASM

```

1 # Escrever um numero por extenso
2 .loadlib 'io_ops'
3
4 print      "Digite um numero de 1 a 5: "
5 read      S1, 2
6 set       I1, S1
7
8 eq        I1, 1, UM
9 eq        I1, 2, DOIS
10 eq       I1, 3, TRES
11 eq       I1, 4, QUATRO
12 eq       I1, 5, CINCO
13
14 print     "Numero invalido!!!"
15 branch   FIM
16
17 CINCO:
18 print     "Cinco"
19 branch   FIM
20
21 QUATRO:
22 print     "Quatro"
23 branch   FIM
24
25 TRES:
26 print     "Tres"
27 branch   FIM
28
29 DOIS:
30 print     "Dois"
31 branch   FIM
32
33 UM:
34 print     "Um"
35
36 FIM:
37 print     "\n"

```

Micro 07

Listagem 3.37: Programa micro 07 em Lua

```

1 -- Listagem 19: Decide se os números são positivos, zeros ou negativos
2
3 --[[ Função: Faça um algoritmo que receba N números e mostre positivo,
   negativo ou zero para cada número --]]
4
5 programa = 1
6 while (programa == 1)
7 do
8   print("Digite um numero: ")
9   numero = io.read()
10  numero = tonumber(numero)
11
12  if(numero > 0)
13  then print("Positivo")
14  elseif(numero == 0)
15  then print("O número é igual a 0")
16  elseif(numero < 0)
17  then print("Negativo")
18  end
19
20
21  print("Deseja Finalizar? (S/N) ")
22  opc = io.read("*line")
23
24  if(opc == "S")
25  then programa = 0
26  end
27 end

```

Listagem 3.38: Programa Micro 07 em PASM

```

1 # Decide se os numeros sao positivos, zeros ou negativos
2 .loadlib 'io_ops'
3
4 LOOP:
5 print      "Digite um numero: "
6 read      S1, 3
7 set       I1, S1
8
9 # Testar se e maior que 0
10 gt        I1, 0, POSITIVO
11 eq        I1, 0, ZERO
12 lt        I1, 0, NEGATIVO
13
14 POSITIVO:
15 print      "Positivo!\n"
16 branch    FINALIZAR
17
18 ZERO:
19 print      "Zero!\n"
20 branch    FINALIZAR
21

```


3.0

```
22 NEGATIVO:
23 print      "Negativo!\n"
24
25 # Parte de DESEJA FINALIZAR?
26 FINALIZAR:
27 print      "Deseja finalizar? (S/N): "
28 read      S10, 2
29 eq        S10, "S\n", FIM
30 branch    LOOP
31
32 FIM:
33 end
```

Micro 08

Listagem 3.39: Programa micro 08 em Lua

```
1 -- Listagem 20: Decide se um numero e maior ou menor que 10
2
3 numero = 1
4 while(numero ~= 0)
5 do
6   print("Escreva um numero: ")
7   numero = tonumber(io.read())
8
9   if(numero > 10)
10  then print("O numero",numero,"e maior que 10")
11  else print("O numero",numero,"e menor que 10")
12  end
13 end
```

Micro 09

Listagem 3.40: Programa micro 09 em Lua

```
1 -- Listagem 21: Calculo de Precos
2
3 print("Digite o preco: ")
4 preco = tonumber(io.read())
5 print("Digite a venda: ")
6 venda = tonumber(io.read())
7
8 if ((venda < 500) or (preco < 30))
9 then novo_preco = preco + (10/100 * preco)
10 elseif ((venda >= 500 and venda < 1200) or (preco >= 30 and preco < 80))
11 then novo_preco = preco + (15/100 * preco)
12 elseif (venda >= 1200 or preco >= 80)
13 then novo_preco = preco - (20/100 * preco)
14 end
15
16 print("O novo preco e: ", novo_preco)
```

Micro 10

Listagem 3.41: Programa micro 10 em Lua

```
1 --Listagem 22: Calcula o fatorial de um numero
```

```

2
3 --[[ Função: recebe um número e calcula recursivamente o fatorial desse nú
      mero --]]
4
5 function fatorial(n)
6     if(n <= 0)
7     then return 1
8     else return (n* fatorial(n-1))
9     end
10 end
11
12 print("Digite um numero: ")
13 numero = tonumber(io.read())
14 fat = fatorial(numero)
15
16 print("O fatorial de", numero, "é: ", fat)

```

Micro 11

Listagem 3.42: Programa micro 11 em Lua

```

1 -- Listagem 23: Decide se um número é positivo, zero ou negativo com o
      auxílio de uma função.
2
3 --[[ Função: recebe um número e verifica se o número é positivo, nulo ou
      negativo com o auxílio de uma função --]]
4
5 function verifica(n)
6     if(n > 0)
7     then res = 1
8     elseif (n < 0)
9     then res = -1
10    else res = 0
11    end
12
13    return res
14 end
15
16 print("Escreva um numero: ")
17 numero = tonumber(io.read())
18 x = verifica(numero)
19
20 if(x==1)
21 then print("Numero positivo")
22 elseif(x==0)
23 then print("Zero")
24 else print("Numero negativo")
25 end

```

Capítulo 4

Referências

- [1] Documentação Lua - <https://www.lua.org/docs.html>
- [2] Documentação OCaml - <https://ocaml.org/docs/>
- [3] Wikibooks, Parrot Virtual Machine - <https://en.wikibooks.org/wiki/Parrot_{VirtualMachine}>
- [4] Wikibooks, PASM Reference - <https://en.wikibooks.org/wiki/Parrot_{VirtualMachine}/PASM_{Reference}>
- [5] Wikibooks, Parrot Intermediate Representation (PIR) - <https://en.wikibooks.org/wiki/Parrot_{VirtualMachine}/PIR>
- [6] - Cardinal, Github - <https://github.com/parrot/cardinal/>
- [7] - Opcodes de PASM - <http://docs.parrot.org/parrot/latest/html/ops.html>
- [8] Parrot Documentation, Exemplos de PASM - <http://parrot.org/dev/examples/pasm>