

# 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

15 de agosto de 2016

# Lista de Figuras

2.1	Instalando e testando LUA . . . . .	7
2.2	Instalando e testando OCaml . . . . .	8
2.3	Instalando e testando Parrot . . . . .	9

# Lista de Tabelas

# Lista de Listagens

2.1	Output Simples em Parrot Assembly Language . . . . .	9
2.2	Output Simples em Parrot Intermediate Representation . . . . .	9
3.1	Programa nano 01 em Lua . . . . .	11
4.1	Nano 02 . . . . .	12
4.2	Nano 03 . . . . .	12
4.3	Nano 04 . . . . .	12
4.4	Nano 05 . . . . .	12
4.5	Nano 06 . . . . .	12
4.6	Nano 07 . . . . .	13
4.7	Nano 08 . . . . .	13
4.8	Nano 09 . . . . .	13
4.9	Nano 10 . . . . .	13
4.10	Nano 11 . . . . .	13
4.11	Nano 12 . . . . .	14
4.12	Micro 01 . . . . .	14
4.13	Micro 02 . . . . .	14
4.14	Micro 03 . . . . .	14
4.15	Micro 04 . . . . .	15
4.16	Micro 05 . . . . .	15
4.17	Micro 06 . . . . .	16
4.18	Micro 07 . . . . .	16
4.19	Micro 08 . . . . .	17
4.20	Micro 09 . . . . .	17
4.21	Micro 10 . . . . .	17
4.22	Micro 11 . . . . .	17

# Sumário

<b>Lista de Figuras</b>	<b>2</b>
<b>Lista de Tabelas</b>	<b>3</b>
<b>1 Introdução</b>	<b>6</b>
<b>2 Instalação dos componentes</b>	<b>7</b>
2.1 Homebrew . . . . .	7
2.2 Lua . . . . .	7
2.2.1 Instalação e Teste . . . . .	7
2.2.2 Informações sobre a linguagem Lua . . . . .	8
2.3 Ocaml . . . . .	8
2.3.1 Instalação e Teste . . . . .	8
2.3.2 Informações sobre a linguagem OCaml . . . . .	8
2.4 Parrot Virtual Machine . . . . .	8
2.4.1 Instalação e Teste . . . . .	8
2.4.2 Informações sobre a Parrot Virtual Machine . . . . .	9
2.4.3 Parrot Assembly Language (PASM) . . . . .	10
2.4.4 Parrot Intermediate Representation (PIR) . . . . .	10
<b>3 Estudando a geração de PIR por um compilador</b>	<b>11</b>
3.1 Compilador e Utilização . . . . .	11
3.1.1 Nano Programas . . . . .	11
<b>4 Programas na Linguagem Lua</b>	<b>12</b>
4.1 Nano Programas . . . . .	12
4.2 Micro Programas . . . . .	14
<b>5 Programas em PASM (Parrot Assembly Language)</b>	<b>19</b>
<b>6 Referências</b>	<b>20</b>

# 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
$ /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
> ^C
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

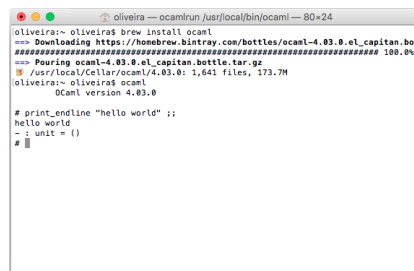
### 2.3.1 Instalação e Teste

Novamente através do homebrew, basta digitar:

```
\$ brew install ocaml
```

Resultado:

**Figura 2.2:** *Instalando e testando OCaml*



### 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 ~ - ssh - 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.o1_capitan.bot
##### 100.0%
=> Pouring parrot-8.1.0.o1_capitan.bottle.tar.gz
=> /usr/local/Cellar/parrot/8.1.0: 700 files, 20.4M
oliveira:~ oliveira$ parrot
parrot -[acEhrtVwy.] -[Dot][MEXFLAGS] [-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 fácil 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].

### 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

# Estudando a geração de PIR por um compilador

### 3.1 Compilador e Utilização

O compilador que será utilizado será o Cardinal [6], é um compilador da linguagem Ruby para a máquina virtual Parrot capaz de gerar código o código intermediário (PIR) como saída.

A documentação do compilador é simples e clara, para baixar o compilador basta digitar no terminal:

```
\$ git clone git://github.com/parrot/cardinal.git
```

Entre as várias opções de instalação, é possível fazê-la utilizando do próprio parrot, para isso basta entrar na pasta onde foi baixado o Cardinal e digitar:

```
\$ winxed setup.winxed build
```

Para compilar é necessário estar na pasta de instalação e o comando é:

```
\$ parrot cardinal.pbc [arquivo].rb
```

Sendo o arquivo o diretório do arquivo Ruby que se deseja executar, para gerar o PIR o comando é:

```
\$ parrot cardinal.pbc -o [output].pir --target=pir [arquivo].rb
```

Sendo output o diretório onde será salvo o arquivo PIR.

#### 3.1.1 Nano Programas

Listagem 3.1: Programa nano 01 em Lua

```
1 -- Listagem 1: Módulo mínimo que caracteriza um programa
```

# Capítulo 4

## Programas na Linguagem Lua

### 4.1 Nano Programas

#### Listagem 4.1: Nano 02

```
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 4.2: Nano 03

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

#### Listagem 4.3: Nano 04

```
1 -- Atribuic a o de uma soma de inteiros a uma varia vel
2 n = 1 + 2
```

#### Listagem 4.4: Nano 05

```
1 -- Inclusa o do comando de impressa o
2 n = 2
3 print(n)
```

#### Listagem 4.5: Nano 06

```
1 -- Listagem 6: Atribuic a o de uma subtrac a o de inteiros a uma
    varia vel
2
3 n = 1 - 2
4 print(n)
```

## Listagem 4.6: Nano 07

```

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

```

---

## Listagem 4.7: Nano 08

```

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 4.8: Nano 09

```

1 -- Listagem 9: Atribuic a o de duas operac o es aritmeticas sobre
   inteiros a uma varia vel
2
3 n = 1 + 1 / 2
4 if (n == 1)
5 then
6   print(n)
7 else
8   print("0")
9 end

```

---

## Listagem 4.9: Nano 10

```

1 -- Listagem 10: Atribuic a o de duas varia veis 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 4.10: Nano 11

```

1 -- Listagem 11: Introduc a o do comando de repetic 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 4.11: Nano 12

```

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

```

## 4.2 Micro Programas

## Listagem 4.12: Micro 01

```

1 -- Listagem 13: Converte graus Celsius para Fahrenheit
2
3 -- [[ Função: Ler uma temperatura em graus Celsius e apresenta-la
    convertida em graus Fahrenheit. A fórmula de conversão 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 4.13: Micro 02

```

1 -- Listagem 14: Ler dois inteiros e decide qual é maior
2
3 --[[ Função : Escrever um algoritmo que leia dois valores inteiro
    distintos e informe qual é 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 4.14: Micro 03

## 4.2

```
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 4.15: Micro 04

```
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 4.16: Micro 05

```
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 4.17: Micro 06

```

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 4.18: Micro 07

```

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 4.19: Micro 08

```

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

```

---

## Listagem 4.20: Micro 09

```

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)

```

---

## Listagem 4.21: Micro 10

```

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, "e: ", fat)

```

---

## Listagem 4.22: Micro 11

```

1 -- Listagem 23: Decide se um número é positivo, zero ou negativo com o
   auxilio 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 5

# Programas em PASM (Parrot Assembly Language)

# Capítulo 6

## Referências

- [1] Documentação Lua - <https://www.lua.org/docs.html>  
Documentação OCaml - <https://ocaml.org/docs/>  
Wikibooks, Parrot Virtual Machine - [https://en.wikibooks.org/wiki/Parrot\\_virtual\\_machine](https://en.wikibooks.org/wiki/Parrot_virtual_machine)  
*Wikibooks, PASM Reference*—[https://en.wikibooks.org/wiki/Parrot\\_virtual\\_machine/PASM\\_reference](https://en.wikibooks.org/wiki/Parrot_virtual_machine/PASM_reference)  
*Wikibooks, Parrot Intermediate Representation (PIR)*—[https://en.wikibooks.org/wiki/Parrot\\_virtual\\_machine/PIR](https://en.wikibooks.org/wiki/Parrot_virtual_machine/PIR)  
— *Cardinal, Github* — <https://github.com/parrot/cardinal/>