Bibliografia

[1] DE FIGUEIREDO, L. H.; IERUSALIMSCHY, R. ; CELES, W.. **Lua — An Extensible Embedded Language**. Dr. Dobb's Journal, 21(12):26–33, 1996.

[2] IERUSALIMSCHY, R.; DE FIGUEIREDO, L. H. ; CELES, W.. **Lua — An Extensible Extension Language**. Software: Practice and Experience, 26(6):635–652, 1996.

[3] OUSTERHOUT, J.. **Scripting: Higher Level Programming for the 21st Century**. IEEE Computer, 31(3):23–30, 1998.

[4] CERQUEIRA, R.; CASSINO, C. ; IERUSALIMSCHY, R.. **Dynamic Component Gluing Across Different Componentware Systems**. In: INTERNATIONAL SYMPOSIUM ON DISTRIBUTED OBJECTS AND APPLICATIONS (DOA'99), 1999.

[5] CASSINO, C.; IERUSALIMSCHY, R.. **LuaJava — Uma Ferramenta de Scripting para Java**. In: SIMPóSIO BRASILEIRO DE LINGUAGENS DE PROGRAMAçãO (SBLP'99), 1999.

[6] CASSINO, C.; IERUSALIMSCHY, R. ; RODRIGUEZ, N.. **LuaJava — A Scripting Tool for Java**. Technical report, Computer Science Department, PUC-Rio, 1999. Available at `http://www.tecgraf.puc-rio.br/~cassino/luajava/index.html`.

[7] ACTIVESTATE. **Release Information for the ActiveState Perl for .NET compiler**, 2000. Available at `http://www.activestate.com/Corporate/Initiatives/NET/Perl_release.html`.

[8] HAMMOND, M.. **Python for .NET: Lessons Learned**, 2000. Available at `http://www.activestate.com/Corporate/Initiatives/NET/Python_for_.NET_whitepaper.pdf`.

[9] INC., S.. **S#.NET Tech-preview Software Release**, 2000. Available at `http://www.smallscript.com/Community/calendar_home.asp`.

PUC-Rio - Certificação Digital Nº 0210489/CA

[10] MASCARENHAS, F.. **LuaInterface: User's Guide**. Computer Science Department, PUC-Rio, 2000. Available at http://www.inf.puc-rio.br/~mascarenhas/luainterface/manual-en.pdf.

[11] CERQUEIRA, R.; NOGUEIRA, L. ; DE MOURA, A. L.. **The LuaOrb Manual**. TeCGraf Computer Science Department, PUC-Rio, 2000. Available at http://www.tecgraf.puc-rio.br/luaorb/.

[12] MEIJER, E.; GOUGH, J.. **Technical Overview of the Common Language Runtime**. Technical report, Microsoft Research, 2002. Available at http://research.microsoft.com/~emeijer/Papers/CLR.pdf.

[13] GOUGH, J.. **Compiling for the .NET Common Language Runtime**. Prentice Hall, 2002.

[14] MICROSOFT. **ECMA C# and Common Language Infrastructure Standards**, 2002. Available at http://msdn.microsoft.com/net/ecma/.

[15] STUTZ, D.. **The Microsoft Shared Source CLI Implementation**, 2002. Available at http://msdn.microsoft.com/library/en-us/Dndotnet/html/mssharsourcecli.asp.

[16] DUBOIS, J.. **PerlNET — The Camel Talks .NET**. In: THE PERL 6 CONFERENCE, 2002. Available at http://conferences.oreillynet.com/presentations/os2002/dubois_update.ppt.

[17] XIMIAN. **The Mono Project**, 2003. Available at http://www.go-mono.com/.

[18] BOCK, J.. **.NET Languages**, 2003. Available at http://www.jasonbock.net/dotnetlanguages.html.

[19] IERUSALIMSCHY, R.. **Programming in Lua**. Lua.org, 2003.

[20] IERUSALIMSCHY, R.; DE FIGUEIREDO, L. H. ; CELES, W.. **Lua 5.0 Reference Manual**. Technical Report 14/03, PUC-Rio, 2003. Available at http://www.lua.org.

[21] SHANKAR, A.. **Implementing Coroutines for .NET by Wrapping the Unmanaged Fiber API**. MSDN Magazine, 18(9), 2003. Available at http://msdn.microsoft.com/msdnmag/issues/03/09/CoroutinesinNET/default.aspx.

PUC-Rio - Certificação Digital Nº 0210489/CA

[22] JENSEN, J.. **LuaPlus 5.0 Distribution**, 2003. Available at `http://wwhiz.com/LuaPlus/index.html`.

[23] MICROSOFT. **Managed Extensions for C++ Migration Guide: Platform Invocation Services**, 2003. Available at `http://msdn.microsoft.com/library/en-us/vcmxspec/html/vcmg_PlatformInvocationServices.asp`.

[24] **Lua: user projects**, 2004. Available at `http://www.lua.org/uses.html`.

[25] MICROSOFT. **JScript .NET Language Reference**, 2004. Available at `http://msdn.microsoft.com/library/en-us/jscript7/html/jsoriprogrammingwithjscriptnet.asp`.

[26] BAGLEY, D.. **The Great Computer Language Shootout**, 2004. Available at `http://dada.perl.it/shootout/`.

PUC-Rio - Certificação Digital Nº 0210489/CA

A
Instruções da Máquina Virtual de Lua 5.0

Neste apêndice estão listadas as 35 instruções da máquina virtual de Lua 5.0. Instruções similares estão agrupadas. Algumas instruções têm argumentos que podem fazer referência tanto a um registrador quanto a uma constante. Se o valor do argumento é menor do que o número máximo de registradores, ele faz referência a um registrador. Se o valor do argumento é maior do que o número máximo de registradores subtrai-se este número do valor; o número obtido é posição da constante, no vetor de constantes, para o qual ele faz referência.

OP_MOVE A B Copia o valor do registrador B para o registrador A.

OP_LOADK A B Copia para o registrador A a constante armazenada na posição B do vetor de constantes.

OP_LOADBOOL A B C Armazena o valor `true` no registrador A se B for diferente de 0, senão armazena o valor `false`. Salta a próxima instrução se C for diferente de 1.

OP_LOADNIL A B Armazena o valor `nil` em todos os registradores entre o registrador A e o B (inclusive).

OP_GETUPVAL A B Copia para o registrador A o upvalue armazenado na posição B do vetor de upvalues.

OP_GETGLOBAL A B Copia para o registrador A o valor da tabela de globais cuja chave é o valor do registrador/constante B.
Se o valor da tabela for `nil` e a tabela possuir o meta-método `__index` chama o meta-método; o valor de retorno do meta-método é copiado para o registrador A.

OP_GETTABLE A B C Copia para o registrador A o valor da tabela armazenada no registrador B cuja chave é o valor registrador/constante C.
Se o valor da tabela for `nil` e a tabela possuir o meta-método `__index`

PUC-Rio - Certificação Digital Nº 0210489/CA

chama o meta-método; o valor de retorno do meta-método é copiado para o registrador A.

OP_SETGLOBAL A B Armazena o valor do registrador A na tabela de globais, usando o valor do registrador/constante B como chave.
Se a chave não existia e a tabela possui o meta-método `__newindex` chama o meta-método.

OP_SETUPVAL A B Copia o valor do registrador A para o upvalue armazenado na posição B do vetor de upvalues;

OP_SETTABLE A B C Armazena o valor do registrador/constante C na tabela armazenada no registrador A, com o registrador/constante C como chave.
Se a chave não existia e a tabela possui o meta-método `__newindex` chama o meta-método.

OP_NEWTABLE A B C Cria uma nova tabela e a armazena no registrador A, usando os valores dos registradores B e C como parâmetros para o tamanho inicial da tabela.

OP_SELF A B C Usado na chamada a métodos, copia o valor do registrador B para o registrador A+1, depois faz o equivalente a OP_GETTABLE A B C.

OP_ADD/OP_SUB/OP_MUL/OP_DIV/OP_POW A B C
Operações aritméticas binárias. Soma/subtrai/multiplica/divide/eleva à potência os valores dos registradores/constantes B e C, armazenando o resultado no registrador A.
Se um dos operandos não for numérico e possuir o meta-método apropriado (`__add`, `__sub`,`__mul`,`__div` ou `__pow`) chama o meta-método e copia seu valor de retorno para o registrador A.

OP_UNM A B Armazena a negação do valor do registrador B no registrador A.
Se o valor do registrador B não for numérico e possuir o meta-método `__unm` chama o meta-método e copia o valor de retorno para o registrador A.

OP_NOT A B Armazena a negação booleana do valor do registrador B no registrador A (`true` se o valor for `nil` ou `false` e `false` em caso contrário).

PUC-Rio - Certificação Digital Nº 0210489/CA

OP_CONCAT A B C Concatena os valores de todos os registradores entre B e C, inclusive, e armazena o resultado no registrador A.
Dois a dois os operandos são verificados; caso um dos dois não seja um número ou string e possua o meta-método `__concat` chama o meta-método com os dois operandos e o valor de retorno é usado.

OP_JMP A Salta A instruções para frente, se A for positivo, ou para trás, se for negativo.

OP_EQ/OP_LT/OP_LE A B C Se A for 0, e o valor do registrador/constante B for igual (para OP_EQ), menor (para OP_LT) ou menor ou igual (para OP_LE) ao valor do registrador/constante C, salta a próxima instrução.
Se A for 1, e o valor do registrador/constante B for diferente (para OP_EQ), maior ou igual (para OP_LT) ou maior (para OP_LE) que o valor do registrador/constante C, salta para a próxima instrução.
Caso os operandos não sejam números ou strings, tenham ambos o mesmo tipo e o mesmo meta-método apropriado para a operação (`__eq`, `__lt` ou `__le`) o meta-método é chamado para fazer a operação de comparação.

OP_TEST A B C Se C for 0, e o valor do registrador B for `nil` ou `false`, faz o equivalente a um OP_MOVE A B. Se C for 1, e o valor do registrador B não for `nil` nem `false`, também faz um equivalente a um OP_MOVE A B. Nos outros casos salta a próxima instrução.

OP_CALL A B C Executa a função armazenada no registrador A. A lista de argumentos começa com o valor do registrador A+1 e vai até o registrador A+B-1. Se B for igual a 1 a função é chamada sem argumentos, se for 0 a lista de argumentos vai do registrador A+1 até o topo da pilha (para o caso em que os argumentos são os valores de retorno de outra função).
O registrador C é o número de valores de retorno desejado menos um. Se C for 0 todos os valores de retorno são aproveitados, e o último valor retornado passa a ser o topo da pilha (para o caso em que os valores de retorno são usados como argumentos para outra função).
Na execução desta instrução a máquina virtual cria um novo registro de ativação para a função e guarda nele o endereço para retorno. O primeiro argumento passa a ser o registrador 0 da nova função.
Se o valor no registrador A não for uma função e possuir o meta-método `__call` todos os argumentos são movidos um registrador para cima; o valor do registrador A é copiado para o registrador A+1; o meta-método é copiado para o registrador A e chamado.

PUC-Rio - Certificação Digital Nº 0210489/CA

OP_TAILCALL A B C Implementa uma chamada a uma função com o retorno de todos os valores retornados por ela (ex. `return func(a,b,c)`). Funciona como OP_CALL A B C, mas reaproveita o registro de ativação e os registradores (o espaço na pilha de execução) da função atual. Os argumentos são copiados para os registradores, a partir do 0. Como o endereço de retorno da nova função passa a ser o da função atual, esta instrução equivale a retornar da função. O retorno implica no abandono do escopo definido pela função, logo os upvalues em aberto desse escopo são fechados.

OP_RETURN A B Retorna da função atual, com o primeiro valor de retorno no registrador A e os outros B-2 valores de retorno nos registradores seguintes. Se B for 1 a função não retorna nenhum valor, e se B for 0 ela retorna todos os valores entre o registrador A e o topo da pilha.
Os valores são copiados para os registradores da função para a qual se está retornando, a partir do que armazena a função chamada. Os valores a mais são descartados e os valores a menos completados com `nil`. Como o escopo definido pela função é abandonado os upvalues em aberto daquele escopo são fechados.

OP_FORLOOP A B A linguagem Lua tem dois tipos de laços *for*. O primeiro é o laço *for* numérico, cuja sintaxe é

```
for idx=inic,lim,incr do
  <bloco>
end
```

onde `idx` é a variável de controle do laço (local a ele), `inic` é o valor inicial, `lim` o limite e `incr` o incremento do a cada iteração.
A instrução OP_FORLOOP é o teste do laço *for* numérico. O registrador A contém a variável de controle, o registrador A+1 o limite do laço e o registrador A+2 o valor de incremento a cada iteração. OP_FORLOOP incrementa o índice e salta B instruções se ele ainda não atingiu o limite.

OP_TFORLOOP A B O segundo tipo da laço *for* é o laço com função de iteração, de sintaxe

```
for idx1,...,idxn in iter,estado,inic do
  <bloco>
end
```

PUC-Rio - Certificação Digital Nº 0210489/CA

onde `idxk` são as variáveis de iteração, `iter` é a função de iteração, `estado` é o valor que é sempre passado como primeiro argumento para `iter` e `inic` o valor inicial para a primeira variável de iteração.
Em cada iteração Lua chama a função de iteração passando o estado e a primeira variável de iteração. Os valores de retorno são armazenados nas variáveis de iteração. Se a primeira variável for `nil` o laço termina.
A instrução OP_TFORLOOP é o teste do laço *for* com função de iteração. O registrador A contém a função de iteração, os registrador A+1 contém o estado e o registrador A+2 a primeira variável de iteração. C é o número de valores de retorno da função de iteração.
OP_TFORLOOP executa a função de iteração e copia seus valores de retorno para os registradores a partir do A+2. Se o registrador A+2 for `nil` salta uma instrução (a instrução seguinte à OP_TFORLOOP é um salto para executar a próxima iteração do laço);

OP_TFORPREP A B Para compatibilidade com as versões anteriores de Lua, Lua 5.0 tem uma terceira sintaxe para laços *for*, específica para iteração em tabelas:

```
for chave,valor in tab do
  <bloco>
end
```

onde `tab` é a tabela que se quer iterar, `chave` e `valor` são variáveis locais ao laço, e que a cada iteração receberão um dos pares de chave e valor presentes na tabela.
OP_TFORPREP é uma instrução que converte um laço *for* para iteração em tabelas em um laço *for* com função de iteração.
OP_TFORPREP copia a tabela armazenada no registrador A para o registrador A+1, então copia a função de iteração `next`, armazenada em uma variável global, para o registrador A e salta B instruções (para uma instrução OP_TFORLOOP).
A tabela passa a ser o estado e o valor inicial é `nil`. `chave` e `valor` passam a ser as variáveis de iteração. A função `next` recebe uma tabela e uma de suas chaves como argumentos e retorna a próxima chave e seu valor.

OP_SETLIST/OP_SETLISTO A B Usados na construção de listas. Os itens que serão inseridos na tabela armazenada no registrador A estão a partir de registrador A+1.

PUC-Rio - Certificação Digital Nº 0210489/CA

No caso de OP_SETLIST, B é o índice do último elemento menos um. O número de itens para inserir é o valor de B módulo uma constante do interpretador, o máximo de itens inseridos de uma só vez, mais um. O índice do primeiro item é o índice do último menos o número de itens para inserir.

OP_SETLISTO é usado quando o último item é uma chamada a uma função. Neste caso, B é o índice do primeiro valor de retorno da função. O número de itens a inserir é o número de elementos entre o registrador A e o topo da pilha. O índice do primeiro item é o valor de B menos B módulo o máximo de itens inserido de uma só vez, mais um.

OP_CLOSE A Fecha os upvalues do escopo atual que estão em aberto (todos os upvalues na apontando para valores na pilha acima da posição do registrador A).

OP_CLOSURE A B Instancia a B-ésima (começando em 0) função declarada pela função atual e a armazena no registrador A. Se a função possui upvalues a instrução é seguida por instruções para inicializar o vetor de upvalues, uma instrução para cada upvalue da função.

Estas instruções podem ser OP_MOVE, se o upvalue aponta para uma variável local do escopo atual (no registrador B da instrução OP_MOVE); nesse caso primeiro se procura o upvalue entre os que já estão em aberto, e caso não exista é criado e posto na lista de upvalues em aberto.

As instruções também podem ser OP_GETUPVAL, se o upvalue aponta para uma variável local de um escopo externo à função atual (um dos upvalues da função atual). Neste caso o upvalue na posição B do vetor de upvalues da função atual é copiado para o vetor de upvalues da função instanciada por OP_CLOSURE.

O processamento das instruções para inicialização do vetor de upvalues é parte da execução da instrução OP_CLOSURE, portanto o interpretador salta estas instruções.

PUC-Rio - Certificação Digital Nº 0210489/CA