MAB 471
2011.2

TINY na LVM

http://www.dcc.ufrj.br/~fabiom/comp

Lua Virtual Machine

- Máquina virtual da linguagem Lua
- Lua é uma linguagem funcional imperativa dinamicamente tipada, com algumas características OO
- Assembler da LVM é bastante alto nível, mas uso de registradores torna ele mais parecido com uma máquina real
- Vamos usar um assembler e um interpretador da LVM escrito em Java - assembler.jar e lua.jar
- O assembler transforma código simbólico da LVM em código executável pelo interpretador

Estrutura do Código Assembler

- Funções declaradas com "function <nome>:"
- Depois da declaração de uma função seguem as instruções e labels do seu corpo
- Uma instrução ou label por linha
- Comentários podem vir depois da instrução ou em uma linha separada, e começam com ';'
- A função chamada "main" é especial; ela tem o código principal do programa (como a main de Java)

Exemplo - fatorial

```
function main:
  CLOSURE       R0 fat 1
  SETGLOBAL     R0 fat
  GETGLOBAL     R0 print
  GETGLOBAL     R1 fat
  LOADK         R2 3
  CALL          R1 2 2
  CALL          R0 2 1
  RETURN        R0 1
```

```
function fat:
  EQ            false R0 0
  JMP           LAB0
  LOADK         R1 1
  RETURN        R1 2
  JMP           LAB1
LAB0:
  GETGLOBAL     R1 fat
  SUB           R2 R0 1
  CALL          R1 2 2
  MUL           R1 R0 R1
  RETURN        R1 2
LAB1:
  RETURN        R0 1
```

```
print(fat(3))
```

```
if(x == 0)
  return 1;
else
  return x * fat(x-1);
```

Registradores e Globais

- Dois espaços de nomes: globais e locais (registradores)
- O espaço global é simbólico (nomes)
- O espaço local é numérico, e plano (sem escopo!)
- Os registradores são numerados começando de 0, até um máximo dependendo da implementação (em geral 250)
- Cada vez que uma função é executada ela ganha um conjunto novo de registradores, então eles são mais como a pilha de uma função C
- Os primeiros registradores são reservados para os parâmetros da função

Operações Básicas

- MOVE Rx Ry - copia o valor de Ry para Rx
- ADD Rx Ry Rz - faz Rx = Ry + Rz
  - ADD pode ser SUB/MUL/DIV
  - Ry e Rz podem ser literais, também (ADD R1 R0 2 é R1 = R0+2)
- LOADK Rx <lit> - carrega o literal dado em Rx
  - Pode ser número ou string
- LOADNIL Rx Ry - carregal **nil** (valor equivalente a **null** na LVM) nos registradores Rx a Ry, inclusive
- LOADBOOL Rx true|false 0 - carrega o **true** ou **false** em Rx
- GETGLOBAL Rx <nome> - carrega o valor de global **<nome>** em Rx
- SETGLOBAL Rx <nome> - escreve o valor de Rx na global **<nome>**

Operações Relacionais e Saltos

- JMP <label> - salta para **<label>**
- EQ true|false Rx Ry, JMP <label> - uma instrução EQ sempre é seguida de uma JMP
  - no caso EQ true, salta para **<label>** se Rx == Ry
  - no caso EQ false, salta para **<label>** se Rx != Ry
  - Rx e Ry podem ser literais
  - Instruções similares LT (<), LE (<=)
- TEST Rx true|false, JMP <label>
  - no caso TEST true, salta para **<label>** se Rx não é **false** ou **nil**
  - no caso TEST false, salta para **<label>** se Rx é **false** ou **nil**
- Operações lógicas implementadas em termos dessas

Chamadas de Função/Método

- O protocolo para chamar uma função é armazenar a função em um registrador Rx e os argumentos nos registradores seguintes R(x +1), R(x+2)...
- De onde vem a função? Pode ser criada na hora usando uma operação CLOSURE, carregada de uma global onde foi guardada anteriormente, ou, no caso de métodos, carregada de um objeto usando SELF
- Uma vez que a função ou método esteja em Rx, chame usando CALL:
  - CALL Rx n 2 - chama a função/método em Rx com n-1 argumentos, e retorna o valor de retorno da função em Rx
  - CALL Rx n 1 - chama a função/método em Rx com n-1 argumentos, sem valor de retorno
  - O primeiro argumento de um método sempre é o objeto, SELF já o armazena em R(x+1)

Chamadas de Função/Método - Exemplos

- Função guardada em uma global

```
; foo(3,<exp>)
; resultado de <exp> em R2
GETGLOBAL     R4 foo
LOADK         R5 3
MOVE          R6 R2
CALL          R1 3 2
; resultado em R4
```

- Método de um objeto

```
; obj.foo(3,<exp>)
; obj em R1
; resultado de <exp> em R2
SELF          R4 R1 foo
LOADK         R6 3
MOVE          R7 R2
CALL          R1 4 2
; resultado em R4
```

Vetores

- Usamos tabelas para representar vetores
- **NEWTABLE Rd n 0** cria um vetor com tamanho inicial **n** e o armazena em Rx
- **GETTABLE Rx Ry Rz** acessa a posição indexada por Rz no vetor em Ry e guarda em Rx
  - Rx = Ry[Rz]
  - Rz pode ser um literal
- **SETTABLE Rx Ry Rz** armazena na posição do vetor Rx indexada por Ry o valor de Rz
  - Rx[Ry] = Rz
  - Ry e Rz podem ser literais
- **LEN Rx Ry** guarda em Rx o tamanho do vetor em Ry