# Compiladores, 2024/2025

Especificação da Máquina Virtual S (versão 2)

Fernando Lobo

## 1 Introdução

Este documento descreve a máquina virtual S (versão 2) que deverá ser implementada para a realização do 2º trabalho prático. A parte que é diferente relativamente à versão anterior está realçada a amarelo.

A máquina virtual S (versão 2) é uma máquina de stack, em tudo idêntica à que foi usada no trabalho anterior. Esta nova versão inclui uma *memória* para armazenar as variáveis globais, bem como mais algumas instruções necessárias para alocar posições de memória, fazer leitura e escrita da memória, e ainda instruções que efectuam saltos condicionais e incondicionais.

Para além dos 4 tipos de dados básicos presentes na versão anterior, a máquina virtual passa a ter um tipo adicional chamado NULO, que significa ausência de valor.

De seguida apresenta-se as instruções da máquina virtual e o seu significado.

## 2 Instruções

### Instruções com 1 argumento (5 bytes)

Instruções com 1 argumento ocupam 5 bytes: 1 byte para o *opcode* (Opc) e 4 bytes para o argumento (um inteiro).

| Opc | Nome | Arg. | Descrição |
|---|---|---|---|
| 0 | iconst | inteiro $n$ | *int constant*: empilha o valor $n$ no stack |
| 1 | dconst | inteiro $n$ | *real constant*: empilha a constante que está na posição $n$ da *constant pool* (supostamente um valor real), no stack. |

| | | | |
|---|---|---|---|
| 2 | sconst | inteiro $n$ | *string constant*: empilha a constante que está na posição $n$ da *constant pool* (supostamente uma string), no stack. |
| 41 | jump | inteiro *addr* | *unconditional jump*: actualiza o *instruction pointer* de modo a que a próxima instrução a ser executada seja aquela que se encontra na posição *addr* do array de instruções. |
| 42 | jumpf | inteiro *addr* | *jump if false*: faz pop do stack. Se o valor for `false`, actualiza o *instruction pointer* de modo a que a próxima instrução a ser executada seja aquela que se encontra na posição *addr* do array de instruções. |
| 43 | galloc | inteiro $n$ | *global memory allocation*: aloca $n$ posições num array que permite armazenar variáveis globais. Designemos esse array por *Globals*. Essas $n$ posições de memória ficam inicializadas com o valor NULO. |
| 44 | gload | inteiro $n$ | *global load*: empilha $Globals[addr]$ no stack. |
| 45 | gstore | inteiro $n$ | *global store*: faz pop do stack e guarda o valor em $Globals[addr]$ |

## Instruções sem argumentos (1 byte)

Instruções sem argumentos necessitam apenas de 1 byte para guardar o *opcode* (Opc).

| **Opc** | **Nome** | Descrição |
|---|---|---|
| 3 | iprint | *int print*: faz pop do operando $a$, e escreve o seu valor no ecrã seguido de um caracter de mudança de linha |
| 4 | iuminus | *int unary minus*: faz pop do operando $a$, e empilha $-a$ no stack. |
| 5 | iadd | *int addition*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha $a + b$ no stack. |
| 6 | isub | *int subtraction*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha $a - b$ no stack. |
| 7 | imult | *int multiplication*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha $a * b$ no stack. |
| 8 | idiv | *int division*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha $a/b$ no stack. |
| 9 | imod | *int modulus*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha o resto da divisão de $a$ por $b$ no stack. |
| 10 | ieq | *int equal*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha o valor lógico de $a == b$ no stack. |

| | | |
|---|---|---|
| 11 | ineq | *int not equal*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha o valor lógico de $a \neq b$ no stack. |
| 12 | ilt | *int less than* faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha o valor lógico de $a < b$ no stack. |
| 13 | ileq | *int less or equal* faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha o valor lógico de $a \leq b$ no stack. |
| 14 | itod | *int to real* converte o valor int que está no topo do stack para um valor real |
| 15 | itos | *int to string* converte o valor int que está no topo do stack para uma string. Ex: 53 é convertido para "53" |
| 16 | dprint | equivalente a iprint, mas para um valor double. |
| 17 | duminus | equivalente a iuminus, mas para um valor double. |
| 18 | dadd | equivalente a iadd, mas para valores do tipo double. |
| 19 | dsub | equivalente a isub, mas para valores do tipo double. |
| 20 | dmult | equivalente a imult, mas para valores do tipo double. |
| 21 | ddiv | equivalente a idiv, mas para valores do tipo double. |
| 22 | deq | equivalente a ieq, mas para valores do tipo double. |
| 23 | dneq | equivalente a ineq, mas para valores do tipo double. |
| 24 | dlt | equivalente a ilt, mas para valores do tipo double. |
| 25 | dleq | equivalente a ileq, mas para valores do tipo double. |
| 26 | dtos | equivalente a itos, mas para um valor double. |
| 27 | sprint | equivalente a iprint, mas para uma string. |
| 28 | sconcat | *string concatenation*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$, e empilha $a$ concatenado com $b$ no stack. |
| 29 | seq | equivalente a ieq, mas para strings. |
| 30 | sneq | equivalente a ineq, mas para strings. |
| 31 | tconst | empilha o valor `true`, do tipo boolean, no stack . |
| 32 | fconst | empilha o valor `false`, do tipo boolean, no stack . |
| 33 | bprint | faz pop do stack. Se o valor for `true` escreve no ecrã `verdadeiro` seguido de um caracter de mudança de linha, se o valor for `false` escreve no ecrã `falso` seguido de um caracter de mudança de linha. |
| 34 | beq | equivalente a ieq, mas para valores do tipo boolean. |
| 35 | bneq | equivalente a ineq, mas para valores do tipo boolean. |
| 36 | and | *boolean and*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$ (supostamente ambos do tipo boolean), e empilha o valor lógico $a$ *and* $b$ no stack. |
| 37 | or | *boolean and*: faz pop do operando direito $b$, seguido de pop do operando esquerdo $a$ (supostamente ambos do tipo boolean), e empilha o valor lógico $a$ *or* $b$ no stack. |
| 38 | not | *boolean not*: faz pop do operando direito $a$ (supostamente do tipo boolean), e empilha o valor lógico *not* $a$ no stack. |
| 39 | btos | equivalente a itos, mas para um valor do tipo boolean. Ex: true é convertido para "true". |
| 40 | halt | termina a execução do programa. |

# 3 Estrutura dos bytecodes

A máquina virtual S executa uma sequência/ficheiro de bytecodes. Esses bytecodes codificam a *constant pool*, seguido das instruções da máquina virtual propriamente ditas.

Os primeiros 4 bytes do ficheiro correspondem a um inteiro que indica o número de entradas da *constant pool*. Se esse número for $n$, sabemos que temos de ler $n$ constantes. Cada constante apenas pode ser um double ou uma string.

De modo a distinguir se estamos perante um double ou uma string, usamos o 1 byte adicional. No caso de ser um double necessitaremos de 8 bytes para o representar (com o byte adicional gastamos 9 bytes por cada double). No caso de ser uma string, necessitamos de saber quantos caracteres tem a string. Para tal usamos um inteiro (4 bytes) para representar o tamanho da string, e depois sabemos exactamente quantos caracteres temos de ler. Cada caracter é codificado com 2 bytes.

## Exemplo

O seguinte exemplo mostra os bytes que correspondem às seguintes constantes:

```
0: 2.0              // 01 40 00 00 00 00 00 00 00
1: 3.14159          // 01 40 09 21 F9 F0 1B 86 6E
2: "ria"            // 03 00 00 00 03 00 72 00 69 00 61
3: "maria"          // 03 00 00 00 05 00 6D 00 61 00 72 00 69 00 61
```

Para cada constante, o 1º byte indica se é double (01) ou string (03). No caso de ser double, os 8 bytes seguintes representam esse double. No caso de ser uma string, os próximos 4 bytes indicam-nos o tamanho da string. Por exemplo, para a string "ria", 00 00 00 03 indicam-nos que a string tem tamanho 3. Já a sequência de bytes 00 00 00 05 indicam-nos que a string "maria"tem tamanho 5. Os restantes bytes são a codificação dos caracteres propriamente ditos. Por exemplo, 00 72 codifica o caracter 'r', 00 69 codifica o caracter 'i', e 00 61 codifica o caracter 'a'.