

# Compiladores, 2024/2025

## Especificação da Máquina Virtual S (versão mini)

Fernando Lobo

### 1 Introdução

Este documento descreve a máquina virtual S (versão mini) que deverá ser implementada para a realização do 1º trabalho prático. Esta versão designa-se por *mini* porque em trabalhos subsequentes iremos acrescentar mais funcionalidades à máquina virtual.

A máquina virtual S é uma máquina de stack, em tudo idêntica à que foi ilustrada na aula teórica 12 e na aula prática PL6.

A diferença é que tem um conjunto de instruções mais alargado, tem uma tabela (*constant pool*) para armazenar constantes que não cabem em 4 bytes (strings e números reais), e o stack de execução passa a poder ter valores de vários tipos, a saber: int, double, boolean, string.

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 $n$	<i>int constant</i> : empilha o valor $n$ no stack
1	dconst	inteiro $n$	<i>real constant</i> : empilha a constante que está na posição $n$ da <i>constant pool</i> (supostamente um valor real), no stack.

2	sconst	inteiro $n$	<i>string constant</i> : empilha a constante que está na posição $n$ da <i>constant pool</i> (supostamente uma string), no stack.
---	--------	-------------	---

## Instruções sem argumentos (1 byte)

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

Opc	Nome	Descrição
3	iprint	<i>int print</i> : faz pop do operando $a$ , e escreve o seu valor no ecrã.
4	iuminus	<i>int unary minus</i> : faz pop do operando $a$ , e empilha $-a$ no stack.
5	iadd	<i>int addition</i> : faz pop do operando direito $b$ , seguido de pop do operando esquerdo $a$ , e empilha $a + b$ no stack.
6	isub	<i>int subtraction</i> : faz pop do operando direito $b$ , seguido de pop do operando esquerdo $a$ , e empilha $a - b$ no stack.
7	imult	<i>int multiplication</i> : faz pop do operando direito $b$ , seguido de pop do operando esquerdo $a$ , e empilha $a * b$ no stack.
8	idiv	<i>int division</i> : faz pop do operando direito $b$ , seguido de pop do operando esquerdo $a$ , e empilha $a/b$ no stack.
9	imod	<i>int modulus</i> : 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	<i>int equal</i> : 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	<i>int not equal</i> : 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	<i>int less than</i> : 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	<i>int less or equal</i> : 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	<i>int to real</i> converte o valor int que está no topo do stack para um valor real
15	itos	<i>int to string</i> 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	<i>string concatenation</i> : faz pop do operando direito <i>b</i> , seguido de pop do operando esquerdo <i>a</i> , e empilha <i>a</i> concatenado com <i>b</i> no stack.
29	seq	equivalente a ieq, mas para strings.
30	sneq	equivalente a ineq, mas para strings.
31	tconst	empilha o valor <b>true</b> , do tipo boolean, no stack .
32	fconst	empilha o valor <b>false</b> , do tipo boolean, no stack .
33	bprint	equivalente a iprint, mas para um valor do tipo boolean.
34	beq	equivalente a ieq, mas para valores do tipo boolean.
35	bneq	equivalente a ineq, mas para valores do tipo boolean.
36	and	<i>boolean and</i> : faz pop do operando direito <i>b</i> , seguido de pop do operando esquerdo <i>a</i> (supostamente ambos do tipo boolean), e empilha o valor lógico <i>a and b</i> no stack.
37	or	<i>boolean and</i> : faz pop do operando direito <i>b</i> , seguido de pop do operando esquerdo <i>a</i> (supostamente ambos do tipo boolean), e empilha o valor lógico <i>a or b</i> no stack.
38	not	<i>boolean not</i> : faz pop do operando direito <i>a</i> (supostamente do tipo boolean), e empilha o valor lógico <i>not a</i> 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
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'.