



UNIVERSIDADE DE ÉVORA

LINGUAGENS DE PROGRAMAÇÃO

---

## Relatório do segundo trabalho prático

---

*Autores:*

João MARQUES, 39996

Tiago MARTINHO, 35735

*Docente:*

Teresa GONÇALVES

Junho de 2020

# Índice

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Correções do Trabalho Anterior</b>	<b>2</b>
<b>3</b>	<b>Registos de Ativação</b>	<b>2</b>
<b>4</b>	<b>Estruturas de dados</b>	<b>3</b>
4.1	Memória de Instruções . . . . .	3
4.2	Memória de Execução . . . . .	3
4.3	Stack de Avaliação . . . . .	4
4.4	Gestor de Etiquetas . . . . .	4
<b>5</b>	<b>Funcionamento das Instruções</b>	<b>4</b>
5.1	Add . . . . .	5
5.2	Sub . . . . .	5
5.3	Mult . . . . .	5
5.4	Div . . . . .	5
5.5	Mod . . . . .	5
5.6	Exp . . . . .	5
5.7	Push int . . . . .	6
5.8	Push var . . . . .	6
5.9	Store var . . . . .	6
5.10	Push arg . . . . .	7
5.11	Store arg . . . . .	7
5.12	Set arg . . . . .	7
5.13	Call . . . . .	8
5.14	Locals . . . . .	9
5.15	Return . . . . .	9
5.16	Jump . . . . .	10
5.17	Jeq . . . . .	10
5.18	Jlt . . . . .	10
5.19	Print . . . . .	11
5.20	Print str . . . . .	11
5.21	Print nl . . . . .	11
<b>6</b>	<b>Execução</b>	<b>12</b>
<b>7</b>	<b>Referências</b>	<b>13</b>

# 1 Introdução

No âmbito da unidade curricular de Linguagens de Programação, pretende-se implementar uma máquina **TISC**. Nesta segunda fase do trabalho pretende-se executar a **memória de instruções**, que foi desenvolvida na primeira fase do trabalho.

Este trabalho foi desenvolvido na linguagem de programação *Java* usando as bibliotecas *JLex* e *JCup*.

Esta implementação da máquina **TISC** tem como objectivo simular e funcionar como um interpretador de TISC.

## 2 Correções do Trabalho Anterior

Face às melhorias propostas pela docente, a estrutura das classes foi repensada de modo a **beneficiar** do paradigma *Object Oriented* do *Java*. Deste modo utiliza-se uma interface *Instruction* que define os métodos **partilhados** por todos os tipos de instruções. Estes métodos são: *execute* e *toString*.

## 3 Registos de Ativação

O **registo de ativação** (RA) é implementado usando um *vector* de *Integer's*. Nesta implementação não existe uma separação física entre cada RA.

Cada registo segue o seguinte diagrama:

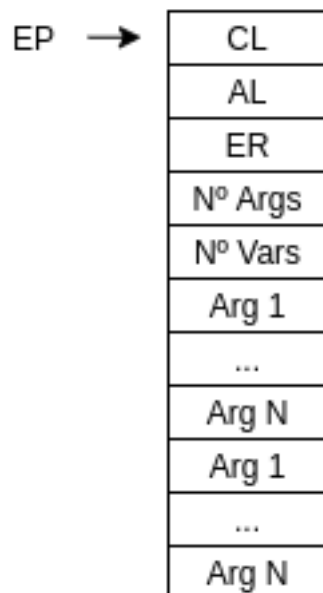


Figura 1: Registo de Ativação

## 4 Estruturas de dados

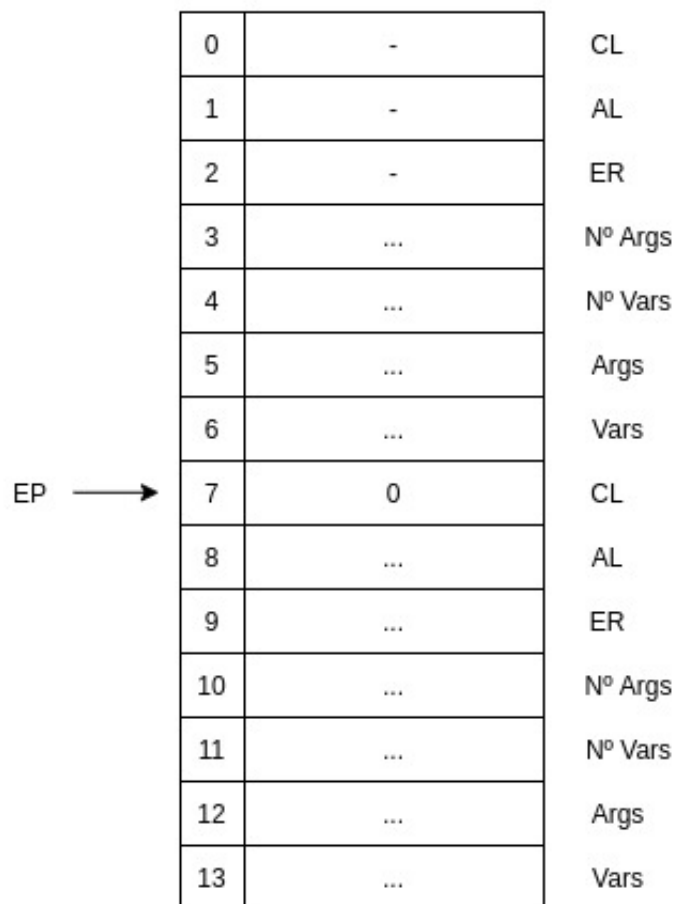
### 4.1 Memória de Instruções

Usou-se a mesma estrutura implementada no trabalho anterior, ou seja, um *vector* de instruções. As instruções são entidades polimórficas sendo que existe uma classe diferente para cada. Estas instruções implementam uma interface que define os métodos que devem existir para cada instrução.

A memória de instruções é iterada usando o PC (program counter) que face às instruções vai variando de valor.

### 4.2 Memória de Execução

A memória de execução é a zona da máquina TISC que vai guardar os registos de ativação. Foi implementada utilizando um *vector* de *Integer* e fazendo uso de um enviroment pointer. É nesta estrutura onde podemos encontrar os registos de ativação atuais, sendo que o enviroment pointer indica o início do ultimo registo.



0	-	CL
1	-	AL
2	-	ER
3	...	N° Args
4	...	N° Vars
5	...	Args
6	...	Vars
EP → 7	0	CL
8	...	AL
9	...	ER
10	...	N° Args
11	...	N° Vars
12	...	Args
13	...	Vars

Figura 2: Memória de Execução

### 4.3 Stack de Avaliação

Foi usado a classe *Stack* encontrada em *Java.util* que estende sobre a classe *Vector* propriedades LIFO (*last in first out*) . Esta stack faz parte das estruturas necessárias para a interpretação da linguagem TISC.

### 4.4 Gestor de Etiquetas

Para gerir as etiquetas utilizou-se um *HashMap* uma vez que se pretende saber o índice da instrução (PC) a qual a etiqueta corresponde.

## 5 Funcionamento das Instruções

- **empilhar** - empilhar um valor na stack de avaliação.
- **desempilhar** - desempilha e devolve o valor da stack de avaliação.
- **registos** - representa o vector memória de execução.
- **FIM** - funciona como o último índice da memória de execução.
- **@** - procura o label no gestor de etiquetas e retorna o índice da instrução referente à memória de instruções.

## 5.1 Add

```
add:
    b = desempilha()
    a = desempilha()
    empilha(a + b)
    PC = PC + 1
```

## 5.2 Sub

```
sub:
    b = desempilha()
    a = desempilha()
    empilha(a - b)
    PC = PC + 1
```

## 5.3 Mult

```
mult:
    b = desempilha()
    a = desempilha()
    empilha(a * b)
    PC = PC + 1
```

## 5.4 Div

```
div:
    b = desempilha()
    a = desempilha()
    empilha(a / b)
    PC = PC + 1
```

## 5.5 Mod

```
mod:
    b = desempilha()
    a = desempilha()
    empilha(a % b)
    PC = PC + 1
```

## 5.6 Exp

```
exp:
    b = desempilha()
    a = desempilha()
    empilha(a ^ b)
    PC = PC + 1
```

## 5.7 Push int

```
push_int (integer1):  
    empilha(integer1)  
    PC = PC + 1
```

## 5.8 Push var

Sendo que *integer1* trata-se da **diferença** entre a profundidade atual e a profundidade onde o bloco que definiu a variável, com a qual queremos interagir, utiliza-se um **ciclo que faz chegar** a esse bloco usando o *access link*. Esta estratégia é usada para aceder a **variáveis** e a **argumentos**.

```
push_var (integer1, integer2):  
    temp = EP  
  
    for i <- 0 to integer1 do  
        AL = temp + 1  
        temp = registros[AL]  
  
    numero_argumentos = registros[temp + 3]  
  
    posicao_variavel = temp + 4 + numero_argumentos + integer2  
  
    empilha(registros[posicao_variavel])  
  
    PC = PC + 1
```

## 5.9 Store var

```
store_var (integer1, integer2):  
    temp = EP  
  
    valor = desempilha()  
  
    for i <- 0 to integer1 do  
        AL = temp + 1  
        temp = registros[AL]  
  
    numero_argumentos = registros[temp + 3]  
  
    posicao_variavel = temp + 4 + numero_argumentos + integer2  
  
    registros[posicao_variavel] = valor  
  
    PC = PC + 1
```

## 5.10 Push arg

```
push_arg (integer1, integer2):  
    temp = EP  
  
    for i <- 0 to integer1 do  
        AL = temp + 1  
        temp = registos[AL]  
  
    posicao_variavel = temp + 4 + integer2  
  
    empilha(registos[posicao_variavel])  
  
    PC = PC + 1
```

## 5.11 Store arg

```
store_arg (integer1, integer2):  
    temp = EP  
  
    valor = desempilha()  
  
    for i <- 0 to integer1 do  
        AL = temp + 1  
        temp = registos[AL]  
  
    posicao_variavel = temp + 4 + integer2  
  
    registos[posicao_variavel] = valor  
  
    PC = PC + 1
```

## 5.12 Set arg

Decidiu-se que a melhor maneira de preparar os argumentos da função chamada seria **introduzir** os seus valores **entre** o registo de ativação atual e o registo de ativação que vai ser criado. Deste modo a próxima função ao executar a instrução *locals* poderá retirar os seus argumentos e guardar no **sítio designado**.

```
set_arg (integer1):  
    valor = desempilha()  
  
    registos[FIM] = valor  
  
    PC = PC + 1
```



## 5.13 Call

Esta instrução trata de criar um **novo registo de ativação** para uma função que vai chamar.

Sabe-se que o *control link* vai guardar o índice do início do registo anterior.

O *access link* é calculado consoante o *integer1*, que vai **representar a distância** entre a profundidade da função chamadora e a da função chamada.

Disto podemos então retirar as seguintes conclusões:

- Se a distância for **menor que zero** então o *access link* é igual ao *control link* do novo bloco.
- Se a distância for **igual a zero**, significa que a nova função se encontra na função anterior pelo que o seu *access link* será igual ao *control link* da função chamadora.
- Se a distância for **maior que zero**, temos então que seguir os *access links* até o sítio em que a função chamada se encontra. Sendo que o *access link* da função chamada será igual ao *control link* do bloco onde a função chamada se encontra.

```
call (integer1, label):
```

```
    // trata do novo EP e guarda o CL para o bloco anterior
```

```
registos[FIM] = EP
```

```
EP_antigo = EP
```

```
EP = FIM
```

```
    // trata do novo AL
```

```
if integer1 < 0 then  
    novo_AL = registos[EP]
```

```
if integer1 = 0 then  
    novo_AL = registos(EP_antigo)
```

```
if integer1 > 0 then  
    EP_anterior = registos(EP_antigo)
```

```
    for i <- 1 to integer1 then  
        novo_AL = registos[EP_anterior + 1]  
        EP_anterior = novo_AL
```

```
registos[FIM] = novo_AL
```

```
PC = @label
```

## 5.14 Locals

Esta instrução vai **reposicionar** os argumentos que a função chamadora guardou entre o seu registo de ativação e o registo da função chamada. Uma vez repostos cria também espaço para **guardar** os valores das **variáveis** declaradas na função.

locals (integer1, integer2):

```
registos[FIM] = integer1
registos[FIM] = integer2

// receber argumentos

inicio_bloco = registos[EP]

// nao valido para a primeira chamada

numero_argumentos_ant = registos[inicio_bloco + 3]
numero_variaveis_ant = registos[inicio_bloco + 4]

argumento = inicio_bloco + numero_argumentos_ant +
            numero_variaveis_ant

for i <- 0 to integer1 do
    registos[FIM] = registos[argumento]
    delete(registos[argumento])
    EP = EP - 1

for int k <- 0 to integer2 do
    registos[k] = NIL

PC = PC + 1
```

## 5.15 Return

return:

```
EP_antigo = EP
PC = registos[EP + 2]
EP = registos[EP]

// apaga registo que ja nao existe

while EP_antigo != FIM + 1 do
    delete(registos[EP_antigo])
```

## 5.16 Jump

jump (label):

```
    PC = @label
```

## 5.17 Jeq

jeq (label):

```
    b = desempilha()
    a = desempilha()
```

```
    if a = b then
        PC = @label
```

```
    else
        PC = PC + 1
```

## 5.18 Jlt

jlt (label):

```
    b = desempilha()
    a = desempilha()
```

```
    if a < b then
        PC = @label
```

```
    else
        PC = PC + 1
```

## 5.19 Print

`print:`

```
a = desempilha()
```

```
print(a)
```

## 5.20 Print str

`print_str (string):`

```
print(string)
```

## 5.21 Print nl

`print_nl:`

```
print(\n)
```

## 6 Execução

Para compilar e executar deve-se utilizar o *Makefile* fazendo *make* e *make run* respectivamente.

Caso se queria ver as mensagens de debug deve-se utilizar os argumentos da função *executa* da máquina TISC. Este debug permite ver as instruções que estão a ser executadas, a memória de execução e a pilha de avaliação.

## 7 Referências

- [1] <https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html>
- [2] <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>