UNIVERSIDADE DE ÉVORA

Execução de programas TISC

Ana Ferro 139872 Eduardo Medeiros 139873

Junho, Ano Letivo 2019/2020

Linguagens de Programação

Prof. Teresa Gonçalves

Índice

1 Registo de Ativação

Figura 1: Desenho do Registo de Ativação (RA)

Este desenho do Registo de Ativação (RA) foi escolhido pelo grupo, pois este permite, de uma maneira fácil e compreensível, efectuar os cálculos necessários para se aceder a uma determinada posição da Memória de Execução a qualquer instante da execução.

2 Estruturas de Dados

Existe uma classe abstrata Instrucao que possui dois métodos abstratos, *executar()* e *toString()*. Todas as classes de instruções da máquina TISC irão estender/ser filhas desta classe, o que fará com que tenham de implementar estes métodos.

Os métodos *toString()* retornam o nome da classe juntamente com as suas variáveis e respetivos valores. Já os métodos *executar()* chamam um método da class TISC que corresponde à respectiva instrução.

2.1 Memória de Instruções

A memória de instruções foi implementada recorrendo a uma ArrayList cujo o tipo é Instrucao. Deste modo quando no analisador sintático é detetada uma nova instrução, seja de que tipo for, esta é adicionada à lista.

Esta adição é feita através do método:

$$public\ void\ adionarInstrucao(Instrucao\ novaInstrucao)$$

Neste método é também incrementada a variável *numeroDeInstrucoes* que será posteriormente usada quando for guardada uma Etiqueta na respetiva memória.

2.2 Memória de Etiquetas

A memória de etiquetas corresponde a uma HashTable em que os tipos da chave e do valor são respetivamente:

$$(Etiqueta,\ Posição\ da\ instrução\ à\ qual\ a\ Etiqueta\ se\ refere)$$

Esta é consultada aquando da execução de instruções que necessitem da posição a que uma determinada Etiqueta se refere.

2.3 Memória de Execução

A memória de execução foi implementada com recurso a uma ArrayList cujo tipo é Integer. Nela vão estar presentes os contéudos dos Registos de Ativação do programa.

Esta é manipulada através das instruções que a máquina executa ao longo da execução do programa.

2.4 Pilha de Avaliação

A Pilha da Avaliação consiste numa Stack cujo o tipo é Integer. Esta é usada na avaliação de expressões, para a transferência de dados e para guardar o valor devolvido por uma função.

2.5 "Memória"de Argumentos

A "Memória"de Argumentos é também implementada com recurso a uma ArrayList cujo tipo é Integer. Esta estrutura serve para guardar o valor dos argumentos, definidos pela instrução *setArg()*, que irão ser usados na chamada da próxima função.

3 Descrição do funcionamento das instruções da máquina TISC

Nota prévia: Após a execução de cada instrução o valor de PC é incrementado em 1. Devido a tal, instruções que alterem o valor do mesmo, deverão alterá-lo para: "novo valor" − 1, uma vez que, assim que a instrução terminar, o valor de PC será incrementado em 1 ficando assim com o valor "novo valor".

3.1 Instruções aritméticas

- `add:`
  ```
  direita = desempilha()
  esquerda = desempilha()
  empilha(esquerda + direita)
  ```

- `sub:`
  ```
  direita = desempilha()
  esquerda = desempilha()
  empilha(esquerda - direita)
  ```

- `mult:`
  ```
  direita = desempilha()
  esquerda = desempilha()
  empilha(esquerda * direita)
  ```

- `div:`
  ```
  direita = desempilha()
  esquerda = desempilha()
  empilha(esquerda / direita)
  ```

- `mod:`
  ```
  direita = desempilha()
  esquerda = desempilha()
  empilha(esquerda % direita)
  ```

- `exp:`
  ```
  expoente = desempilha()
  base = desempilha()
  empilha(base ^ expoente)
  ```

3.2 Instruções para manipulação de inteiros

- pushInt inteiro:
  empilha(inteiro)

3.3 Instruções de acesso a variáveis

- pushVar distancia numero:

```
ra = EVP

while distancia > 0
    ra = getAL(ra)
    distancia = distancia - 1

//Já estando no bloco pretendido,
//aceder ao valor da var na posição numero

empilha(var[numero])
```

- storeVar distancia numero:

```
ra = EVP

while distancia > 0
    ra = getAL(ra)
    distancia = distancia - 1

//Já estando no bloco pretendido,
//aceder ao valor da var na posição numero

var[numero] = desempilha()
```

3.4 Instruções de acesso a argumentos

- pushArg distancia numero:

```
ra = EVP

while distancia > 0
    ra = getAL(ra)
    distancia = distancia - 1

//Já estando no bloco pretendido,
```

```
    //aceder ao valor da var na posição numero

    empilha(arg[numero])
```

- storeArg distancia numero:

```
    ra = EVP

    while distancia > 0
        ra = getAL(ra)
        distancia = distancia - 1

    //Já estando no bloco pretendido,
    //aceder ao valor da var na posição numero

    arg[numero] = desempilha()
```

3.5 Instruções para chamada de funções

- setArg inteiro:

```
    memoriaArgumentos[inteiro] = desempilha()
```

- call distancia etiqueta:

```
    pc1 = pc + 1

    distanciaCall = distancia

    jump etiqueta
```

- locals argumentos variaveis

```
    //Primeiro RA
    if EVP == -1
        EVP = 0

        CL = NIL
        AL = NIL
        ER = NIL

        numeroDeArgumentos = argumentos
        numeroDeVariaveis = variaveis
```

```
    i = 0
    while i < argumentos + variaveis
        reservarEspaco()
        i = i + 1

//Qualquer outro RA
else
    CL = EVP

    if distanciaCall < 0
        AL = EVP
    else
        tempEVP = getAL(EVP)

        while distanciaCall > 0
            tempEVP = getAL(tempEVP)
            distanciaCall = distanciaCall - 1

        AL = tempEVP

    EVP = @CL

    ER = pc1

    numeroDeArgumentos = argumentos
    numeroDeVariaveis = variaveis

    i = 0
    while i < argumentos + variaveis
        reservarEspaco()
        i = i + 1

    if memoriaArgumentos != VAZIO

        i = 0
        while i < memoriaArgumentos.tamanho
            arg[i] = memoriaArgumentos[i]
            i = i + 1

    limpar(memoriaArgumentos)
```

- returnInst

```
if instrucao == ultima
    pc = memoriaDeInstrucoes.tamanho
else
    pc = getER(EVP)

if getCL(EVP) == NIL
    EVP = -1
else
    EVP = getCL(EVP)

destruirRAAtual()
```

3.6 Instruções de salto

- jump etiqueta:

```
pc = @etiqueta - 1
```

- jeq etiqueta:

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

if a == b
    pc = @etiqueta - 1
```

- jlt etiqueta:

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

if a > b
    pc = @etiqueta - 1
```

3.7 Instruções de saída

- print:

```
print(desempilha())
```

- println:

```
print("\n")
```

- `printString string:`
  `print(string)`

4 Compilação e Execução

O programa deve ser compilado através do comando ***make*** e corrido usando ***make run*** $<$ ***exemplos***/ $<$***nome_ficheiro***$>$***.tisc***.