**Execução de Programas CIMS**

Trabalho realizado por:

-Alexandre Rodrigues 35449

-Pedro Oliveira 35480

**Índice**

Introdução......................................................................................................................................3

Registos de Ativação.......................................................................................................................4

Estruturas.......................................................................................................................................5

Implementação da Máquina...........................................................................................................6

Pseudocódigo das Instruções....................................................................................................7/11

Observações.................................................................................................................................12

Conclusão.....................................................................................................................................13

**Introdução**

Foi-nos pedido anteriormente que implementássemos uma máquina abstracta CIMS, cujo objetivo inicial era apenas ler um programa CIMS padrão e carregá-lo na memória de instruções na máquina. Para tal usámos uma ArrayList, por razões especificadas no relatório da da fase anterior.

Nesta segunda fase do trabalho deparámo-nos com alguns obstáculos devido ao Java, portanto decidimos recomeçar o trabalho na linguagem C. A finalidade desta fase é executar o programa através da memória de instruções.

**Registos de Ativação**

Cada registo de ativação está guardado no respetivo bloco. Apesar de nos ter sido pedido para guardar na memória apenas os registos de ativação, decidimos, por questões de otimização, fazer da nossa memória um Array de blocos onde em cada bloco estaria guardado o registo de ativação numa variável.

**Estruturas**

newArray

* Array onde são guardadas apenas as Labels das funções.

newArrayInst

* Array onde são guardadas as instruções.

blocoCoisas

* Array onde são guardados os blocos.

pilha

* Pilha de avaliação.

temp1

* É usado na instrução ‘coloca\_arg’ para posteriormente lhes podermos aceder.

temp2

* É usado na instrução ‘locais’ para atribuir os argumentos à função chamada.

**Implementação da Máquina**

Um bloco é constituído por um array de variáveis, um array de argumentos, duas variáveis para guardar o número de argumentos e o número de variáveis, uma flag para indicar se o bloco ainda está ativo e o registo de ativação.

Inicialmente percorremos o newArray à procura da label ‘Main’, de seguida iremos percorrer o newArrayInst usando o Program Counter, que possuirá o valor da label ‘Main’.

Para cada posição do program counter, usando um switch case, iremos executar cada instrução.

O funcionamento de cada instrução é descrito no próximo capítulo.

**Pseudocódigo das Instruções**

**chama**

for i < newArray.tamanho { if (strcmp(Label.current,Label.requested)) == 0 { pc.new = Label.current.index; } } Ra.new = pc + 1; pc = pc.new – 1;

**locais**

if blocoCoisas.tamanho >= blocoCoisas.espacoAlocado { blocoCoisas.espacoAlocado \* 2; } if bloco.current.flag == 0 { bloco.current.flag = 1; } else { bloco.current.nArgs = locais.inteiro1; bloco.current.nVars = locais.inteiro2; bloco.current.flag = 1; bloco.current.Ra = tempRa; } while(temp1.notEmpty) { push(temp2, temp1.pop); } i = 0; while(temp2.notEmpty) { bloco.current.arrayArgs[i++] = temp2.pop; } blocoCoisas.tamanho++;

**regressa**

if((bloco.current.tamanho - 1).ra == -1) { return; } else { (bloco.current.tamanho - 1).flag = 0; pc = (bloco.current.tamanho - 1).ra -1; blocoCoisas.tamanho--; }

**coloca\_arg**

temp1.push(pilha.pop);

**empilha\_var**

pos = (blocoCoisas.counter - 1) - empilha\_var.inteiro1; pilha.push(blocoCoisas[pos].arrayVars(empilha\_var.inteiro2 - 1));

**empilha\_arg**

pos = (blocoCoisas.counter - 1) - empilha\_var.inteiro1; pilha.push(blocoCoisas[pos].arrayArgs(empilha\_var.inteiro2 - 1));

**empilha**

pilha.push(empilha.inteiro1);

**atribui\_var**

pos = (blocoCoisas.counter - 1) - empilha\_var.inteiro1; val = pilha.pop; blocoCoisas[pos].arrayVars[atribui\_var.inteiro2 - 1] = val;

**atribu\_arg**

pos = (blocoCoisas.counter - 1) - empilha\_arg.inteiro1; val = pilha.pop; blocoCoisas[pos].arrayArgs[atribui\_arg.inteiro2 - 1] = val;

**soma**

y = pilha.pop; x = pilha.pop; x = x + y; pilha.push(x);

**sub\_**

y = pilha.pop; x = pilha.pop; x = x - y; pilha.push(x);

**mult\_**

y = pilha.pop; x = pilha.pop; x = x \* y; pilha.push(x);

**div\_**

y = pilha.pop; x = pilha.pop; w = 0;

if(y == 0) { pilha.push(0); } else { w = x / y; pilha.push(w); }

**exp\_**

y = pilha.pop; x = pilha.pop; w = 0; w = pow(x, y); pilha.push(w);

**mod**

y = pilha.pop; x = pilha.pop; x = x % y; pilha.push(x);

**sigual**

y = pilha.pop; x = pilha.pop;

if(x == y) { Pc.new = 0; i = 0; for(i to newArray.counter, i++) { if(strcmp(newArray.label, sigual.string1) == 0) { Pc.new = newArray.index; } } pc = Pc.new - 2; }

**smenor**

y = pilha.pop; x = pilha.pop; if(x < y) { Pc.new = 0; i = 0; for(i to newArray.counter, i++) { if(strcmp(newArray.label, smenor.string1) == 0) { Pc.new = newArray.index; } } pc = Pc.new - 2; }

**salta**

Pc.new = 0; i = 0; for(i to newArray.counter, i++) { if(strcmp(newArray.label, salta.string1) == 0) { Pc.new = newArray.index; } } pc = Pc.new - 2;

**escreve\_int**

print(pilha.pop);

**escreve\_str**

print(escreve\_int.string1);

**muda\_linha**

print(\n); continue;

**Observações**

Acompanham o enunciado apenas quatro exemplos, **fatorial.cims**, **fatorial\_rec.cims**, **mdc.cims** e **mixtura.cims.** Para correr os ficheiros individualmente basta fazer:

* make runFatorial – corre o programa fatorial.cims.
* make runFatorial\_rec – corre o programa fatorial\_rec.cims.
* make runMdc – corre o programa mdc.cims.
* make runMixtura – corre o programa mixtura.cims.

Para correr todos os ficheiros simultaneamente, usa-se o comando **make runAll.**

**Conclusão**

Depois de superados os obstáculos iniciais, e eliminados os problemas do Java, ao reescrevermos em o trabalho em C, tornou-se mais percetível.