2º Trabalho de Linguagens de Programação

Execução de programas TISC

Docente: Teresa Gonçalves

Alunos:

Miguel Azevedo, nº36975

Vasco Crespo, nº37913

Introdução

O objetivo do primeiro trabalho era fazer uma representação da memória de instruções de um programa TISC apenas representando por que ordem as instruções do programa eram executadas. Neste trabalho o objetivo já não é mostrar a memória de instruções mas sim fazer mesmo uma memória da máquina TISC ou seja, utilizando a pilha de avaliação, a memória de instruções e os registos de ativação, obtendo assim o output de um programa TISC.

Registos de Ativação

Decidimos considerar a nossa memória de execução como sendo uma Stack contendo os registos de ativação. Um registo de ativação é criado quando é lido uma instrução *call* no programa e este registo de ativação estará a apontar para outro registo de ativação que refere onde a função chamada foi criada, no nosso caso este registo de ativação conterá um inteiro da posição na memória de execução onde estará o outro registo de ativação da função onde esta outra função foi criada e também conterá um inteiro que será o *Program_Counter_*antigo de onde aconteceu a chamada da função para que quando esta função termine o programa poder voltar para a posição anterior. Cada registo de ativação irá ter 2 arrays de inteiros, que irão ser as variáveis e os argumentos declarados em cada registo.

Estruturas de Dados utilizadas

A nossa máquina TISC contém:

- As Etiquetas que são representadas por uma *Hashtable* onde em cada posição está uma *String* que contém o nome da respetiva etiqueta e um inteiro que representa a posição em que essa etiqueta se encontra.
- A Memória de Instruções que é uma *ArrayList* que contém as diferentes instruções
- A Pilha de Avaliação que é uma *Stack* que guarda variáveis que serão utilizadas nas diferentes operações.
- Uma lista auxiliar que também é uma *Stack* que apenas é utilizada auxiliarmente em algumas operações.
- Os Registos de Ativação que também são uma *Stack.*

Funcionamento das Instruções

Add:

O2 = desempilha()

O1 = desempilha()

Empilha(o1+o2)

PC = PC +1

Esta instrução desempilha os últimos dois elementos que estão na pilha de avaliação e empilha a soma entre os mesmos.

Sub:

O2 = desempilha()

O1 = desempilha()

Empilha(o1-o2)

PC = PC +1

Esta instrução desempilha os últimos dois elementos que estão na pilha de avaliação e empilha a subtração entre os mesmos.

Mult:

O2 = desempilha()

O1 = desempilha()

Empilha(o1*o2)

PC = PC +1

Esta instrução desempilha os últimos dois elementos que estão na pilha de avaliação e empilha a multiplicação entre os mesmos.

Div:

O2 = desempilha()

O1 = desempilha()

Empilha(o1/o2)

PC = PC +1

Esta instrução desempilha os últimos dois elementos que estão na pilha de avaliação e empilha a divisão entre os mesmos.

Mod:

O2 = desempilha()

O1 = desempilha()

Empilha(o1%o2)

PC = PC +1

Esta instrução desempilha os últimos dois elementos que estão na pilha de avaliação e empilha o resto da divisão inteira entre os mesmos.

Exp:

O2 = desempilha()

O1 = desempilha()

Empilha(o1^o2)

PC = PC +1

Esta instrução desempilha os últimos dois elementos que estão na pilha de avaliação e empilha o primeiro número elevado ao segundo entre os mesmos.

Push_int <inteiro>:

Maquina.pilhadeavaliacao.push(inteiro)

Esta instrução empilha o <inteiro> na pilha de avaliação.

Push_var <inteiro n1> <inteiro n2>:

```
RegistosAtivacao temp = maquina.registosAtivacao.peek();

for (int i = 0; i < n1; i++)
{
    temp = maquina.registosAtivacao.get(temp.acessLink);
}
maquina.getPilha().push(temp.getVariaveis()[n2 - 1]);
maquina.pc++;
```

Esta instrução empilha na pilha de avaliação a variável que está no registo de ativação a uma profundidade <n1>, na posição <n2>.

Para isso, é obtido o registo de ativação que está no topo da memória de execução. Através desse, vamos até ao registo de ativação que se encontra a distância n1 seguindo os acessLinks correspondentes, obtendo assim a variável na posição n2.

Se a distancia for 0 (n1 == 0) significa que o registo de ativação que queremos obter as variáveis é o que está no topo da pilha.

Store_var <inteiro n1> <inteiro n2>:

```
RegistosAtivacao temp = maquina.registosAtivacao.peek();

for (int i = 0; i < n1; i++)
{
    temp = maquina.registosAtivacao.get(temp.acessLink);
}
int topo = maquina.getPilha().pop();
temp.variaveis[n2 - 1] = topo;

maquina.pc++;
```

Esta instrução desempilha o valor que está no topo da pilha de avaliação, e atribui-o à variável identificada pelo <n2> cujo registo de ativação está a uma profundidade <n1>.

Para isso, é obtido o registo de ativação que está no topo da memória de execução. Através desse, vamos até ao registo de ativação que se encontra a distância <n1> seguindo os acessLinks correspondentes, atribuindo assim a variável àquele registo de ativação.

Se a distancia for 0 (n1 == 0) significa que o registo de ativação que queremos obter é o que está no topo da pilha, fazendo assim a atribuição.

Push_arg <inteiro n1> <inteiro n2>:

```
RegistosAtivacao temp = maquina.registosAtivacao.peek();

if(n1 == 0)
    maquina.getPilha().push(temp.getArgs()[n2-1]);
else
    for(int i = 0; i < n1; i++)
        temp = maquina.registosAtivacao.get(temp.acessLink);
    maquina.getPilha().push(temp.getArgs()[n2-1]);
maquina.pc++;
```

Empilha, na pilha de avaliação, o valor do argumento da posição<n2> da função cuja está a profundidade <n1>.

Para isso, é obtido o registo de ativação que está no topo da memória de execução. Através desse, vamos até ao registo de ativação que se encontra a profundidade <n1> seguindo os acessLinks correspondentes, empilhando assim na pilha de avaliação o valor do argumento na posição <n2>.

Se a profundidade for 0 (n1 == 0) significa que o registo de ativação que queremos obter é o que está no topo da pilha, fazendo assim o empilhamento do argumento na posição <n2> daquele registo de ativação.

Store_arg <inteiro n1> <inteiro n2>:

```
RegistosAtivacao temp = maquina.registosAtivacao.peek();

for (int i = 0; i < n1; i++)
    temp = maquina.registosAtivacao.get(temp.acessLink);

int topo = maquina.getPilha().pop();
temp.args[n2 - 1] = topo;
maquina.pc++;
```

Desempilha o valor do topo da pilha de avaliação e atribui-o ao argumento na posição <n2>.

Para isso, é obtido o registo de ativação que está no topo da memória de execução. Através desse, vamos até ao registo de ativação que se encontra a profundidade <n1> seguindo os acessLinks correspondentes, atribuindo assim o valor ao argumentos na posição <n2>.

Se a profundidade for 0 (n1 == 0) significa que o registo de ativação que queremos obter é o que está no topo da pilha, atribuindo assim o valor do topo da pilha de avaliação ao argumento na posição <n2> daquele registo de ativação.

Set_arg<inteiro n1> :

```
maquina.getList().add(n1-1,maquina.getPilha().pop());
maquina.pc++;
```

Esta instrução vai desempilhar o valor <n1> da função que vai ser chamada da nossa pilha de avaliação através do método pop() e vai colocá-lo na posição n1-1 da nossa lista auxiliar através do método add() para que esta função lhe possa aceder.

Call <inteiro n> <String etiqueta>:

```
RegistosAtivacao registo = new RegistosAtivacao();
registo.setProgramCounter_antigo(maquina.pc+1);
registo.nome = etiqueta;
if(n == -1)
{
    registo.acessLink = maquina.registosAtivacao.size()-1;
}
else
{
    RegistosAtivacao temp = maquina.ep;
    for(int i = 0; i < n; i++)          //se for 0 o acess link
    {
        temp = maquina.registosAtivacao.get(temp.acessLink);
    }
    registo.acessLink = temp.acessLink;
}
maquina.registosAtivacao.push(registo);
maquina.ep = registo;
maquina.pc = maquina.getEtiquetas().get(etiqueta);
```

Cada vez que é feita uma instrução *Call* é criado um registo de ativação na qual é guardado o endereço de retorno, para quando a houver uma instrução *return* o p*rogram counter* voltar a apontar para onde estava antes de a função ser chamada.

Caso a distância <n> seja –1, o *acessLink* do novo registo de ativação criado irá ficar a apontar para a posição do registo de ativação que está no topo da nossa memória de instruções.

Caso seja 0, o *acessLink* irá apontar para o mesmo acessLink que o registo de ativação que está no topo da pilha está a apontar.

Caso seja maior do que 0, o processo irá ser o mesmo.

Locals <inteiro n1> <inteiro n2>:

```
RegistosAtivacao registo = maquina.registosAtivacao.peek();
//System.out.println(maquina.getList().toString());
if(n1 > 0)
{
    registo.setArgs(n1);

    for(int i = 0; i < n1; i++)
        registo.args[i] = maquina.getList().get(i);

}
if(n2 > 0)
{
    registo.setVariaveis(n2);
}
maquina.ep = registo;
maquina.pc++;
```

Esta instrução tem como função saber quantos argumentos irá ter a nossa função <n1> e quantas variáveis <n2>.

Para isso, vamos atribuir ao registo de ativação que esta no topo da memória de execução os argumentos, que foram previamente definidos pela instrução *set_arg.* O *<n2>* irá ser quantas variáveis locais irá exisitir naquele registo de ativação. Para isso usamos a instrução *setVariaveis(n2)* que cria um *array* de inteiros com dimensão <n2> associado a cada registo de ativação.

Return:

```
RegistosAtivacao temp = maquina.registosAtivacao.pop();
if(!maquina.registosAtivacao.empty())
{
    maquina.pc = temp.programcounter_antigo++;
    maquina.ep = maquina.registosAtivacao.peek();
}
```

Esta instrução desempilha o último registo de ativação da memória de execução e atualiza o *program counter* para o *endereço de retorno* que foi guardado nesse registo de ativação. O *enviremental pointer* é atualizado para o registo de ativação que está no topo da memoria de execução.

Jump <String etiqueta>:

```
maquina.pc = maquina.getEtiquetas().get(etiqueta);
```

A instrução *Jump* passa o *Program Counter* para a etiqueta com o nome <etiqueta> que passa a ser a próxima instrução a ser executada.

Jeq <String etiqueta>:

```
int o2 = maquina.getPilha().pop();
int o1 = maquina.getPilha().pop();

if(o1 == o2)
    maquina.pc = maquina.getEtiquetas().get(etiqueta);

else
    maquina.pc++;
```

A instrução *Jeq* desempilha as duas últimas posições da nossa Pilha de Avaliação e caso sejam iguais o *Program Counter* passa para a instrução com o nome <etiqueta>. Caso não seja o *Program Counter* passa para a próxima instrução a seguir à instrução jeq.

Jlt <String etiqueta>:

```
int o2 = maquina.getPilha().pop();
int o1 = maquina.getPilha().pop();

if(o1 < o2)
    maquina.pc = maquina.getEtiquetas().get(etiqueta);

else
    maquina.pc++;
```

A instrução *Jlt* desempilha as duas últimas posições da nossa Pilha de Avaliação e caso o valor no topo da pilha seja superior ao valor imediatamente abaixo o *Program Counte*r passa para a instrução com o nome <etiqueta> que passa a ser a próxima instrução a ser executada. Caso não seja, o *Program Counter* passa para a próxima instrução a seguir à instrução *jlt*.

Print:

```
System.out.print(maquina.getPilha().pop());
maquina.pc++;
```

A instrução Print desempilha o último valor da Pilha de Avaliação e mostra esse valor na saída do programa. Aumenta também o valor do Program Counter que passa para a instrução seguinte.

Print_str<String string>:

```
System.out.print(" "+string);
maquina.pc++;
```

A instrução Print_str apenas mostra na saída do programa o que recebe na <string> aumentando também o valor do Program Counter para passar para a instrução seguinte.

Print_nl:

```
System.out.println();
maquina.pc++;
```

A instrução Print_nl apenas termina a linha corrente na saída do programa e aumenta o valor do Program Counter que passa para a próxima instrução.