Linguagens de Programação 2019/2020

Departamento de Informática, Universidade de Évora

1º Trabalho Prático

– Leitura de programas TISC –

1 Objectivo

Utilizando uma linguagem de programação à escolha (Java, Python, C), pretende-se implementar uma máquina **TISC**. Neste primeiro trabalho, a implementação deverá ler um programa TISC da entrada padrão e carregá-lo na memória de instruções.

2 A máquina TISC

TISC (`Tiny Instruction Set Computer`) é uma máquina abstrata simples, preparada para a execução de programas escritos numa linguagem estruturada em blocos, com âmbito de identificadores estático e definição de funções e procedimentos locais.

A máquina TISC tem três zonas de memória distintas:

1. a **memória de instruções**, onde são guardadas as instruções do programa a executar (formato não especificado);
2. a **pilha de avaliação**, usada na avaliação de expressões, para a transferência de dados e para guardar o valor devolvido por uma função
3. a **memória de execução**, onde se encontram os registos de ativação dos blocos do programa cuja execução ainda não terminou.

Além daquelas zonas da memória, a máquina TISC possui dois registos:

1. o registo **EP**, que contém o `environment pointer`
2. o registo **PC**, para o `program counter`

Caso seja considerado necessário, poder-se-ão acrescentar outros registos que ajudem a controlar o funcionamento da máquina; no entanto, estes registos não poderão conter dados manipulados pelo programa.

3 A linguagem iPascal

A linguagem **iPascal** é uma variante da linguagem Pascal que permite manipular inteiros e onde a abertura e fecho de um bloco é identificado pelas palavras `begin` e `end`, respectivamente. Esta linguagem permite definir funções locais a uma função e todas as variáveis são globais no corpo da função onde são declaradas, ou seja, a definição de uma função tem a forma:

```
type name(formal arguments)
begin
  declarations        /* variáveis e funções */
  instructions
end
```

onde `instructions` não inclui qualquer declaração de variável ou função.

Um programa é definido através da função `program()`, como é exemplificado pelo programa seguinte, que calcula e escreve no terminal factorial de 5:

```
program()
begin
  int fact(int n)
  begin
    int i, f = 1;

    for (i = 1; i <= n; ++i)
      f = f * i;
    return f;
  end
  println(fact(5));
end
```

A **profundidade** de um bloco é dada pelo número de blocos que o envolvem no texto do programa. A função `program` é o único bloco com profundidade 0. No exemplo acima, o bloco correspondente à função `fact`, tem profundidade 1. Não há restrição quanto à profundidade máxima de um bloco.

Os programas iPascal são compilados para sequências de instruções TISC e executadas por uma implementação da máquina abstrata. Para o programa anterior, um compilador poderia gerar o seguinte código TISC:

```
program:   locals 0 0
           push_int 5
           set_arg 1
           call -1 fact    # calcula fact(5)
           print           # e imprime
           print_nl
           return

   fact:   locals 1 2
           push_int 1
           store_var 0 2   # f = 1
           push_int 1
           store_var 0 1   # i = 1
   L1:     push_arg 0 1
           push_var 0 1
           jlt L2        # n < i
```

```
        push_var 0 2
        push_var 0 1
        mult
        store_var 0 2   # f = f * i
        push_var 0 1
        push_int 1
        add
        store_var 0 1   # i = i + 1
        jump L1
L2:     push_var 0 2
        return          # return f
```

4 As instruções TISC

Um programa TISC é uma sequência de instruções TISC, onde cada instrução pode ser identificada por uma etiqueta. As instruções são executadas pela ordem em que aparecem no texto do programa, a partir da primeira instrução da função `program`, identificada pela etiqueta `program`. A ordem normal de execução das instruções pode ser alterada com a execução das instruções de salto ou das instruções `call` e `return`.

4.1 Instruções aritméticas

Estas instruções executam as operações aritméticas da máquina TISC. Não possuem argumentos (operam sobre o conteúdo da pilha de avaliação) e funcionam em quatro passos:

1. desempilha o segundo operando da pilha de avaliação
2. desempilha o primeiro operando da pilha de avaliação
3. efetua a operação
4. empilha o resultado na pilha de avaliação

As instruções aritméticas são:

- `add`
  Soma
- `sub`
  Subtração
- `mult`
  Multiplicação
- `div`
  Divisão inteira
- `mod`
  Módulo da divisão inteira
- `exp`
  Exponenciação

4.2 Instruções para manipulação de inteiros

- `push_int <inteiro>`
  Esta instrução empilha `<inteiro>` (uma constante inteira) na pilha de avaliação.

4.3 Instruções de acesso a variáveis

As variáveis são declaradas no início dos blocos que constituem a definição das funções (na declaração não se incluem os argumentos da função). Estas instruções são responsáveis pela leitura e escrita do valor de uma variável:

- `push_var <inteiro1> <inteiro2>`
  Sejam $P0$ a profundidade do bloco em que determinada variável foi declarada e $P1$ a profundidade do bloco em que ela está a ser acedida:
  - `<inteiro1>` é a **distância** entre o bloco corrente e aquele onde a variável foi declarada (diferença entre $P1$ e $P0$). Se a variável é local à função corrente, esse valor é 0; se a variável foi declarada na função envolvente à função corrente esse valor é 1 e assim sucessivamente.
  - `<inteiro2>` é o **número** da variável no bloco em que foi declarada. Em cada bloco as variáveis têm números distintos e consecutivos começando em 1. Nestas condições, um número de uma variável identifica-a univocamente dentro do seu bloco.

  O efeito desta instrução é empilhar, na pilha de avaliação, o valor da variável identificada pelo número `<inteiro2>` do bloco cuja profundidade dista `<inteiro1>` da profundidade do bloco corrente.

- `store_var <inteiro1> <inteiro2>`
  Os argumentos desta instrução são idênticos ao da instrução anterior. Esta instrução desempilha o valor no topo da pilha de avaliação e atribui-o à variável identificada pelo número `<inteiro2>` do bloco cuja profundidade dista `<inteiro1>` da profundidade do bloco corrente.

4.4 Instruções de acesso a argumentos

A função destas instruções TISC é idêntica à do grupo anterior, para os argumentos das funções. O 1º argumento é a distância entre a profundidade da função a que o argumento pertence e a da função que lhe acede e o 2º é o número de ordem do argumento na lista de argumentos formais da função (o primeiro é identificado pelo número 1):

- `push_arg <inteiro1> <inteiro2>`
  Empilha, na pilha de avaliação, o valor do argumento `inteiro2` da função cuja profundidade dista `inteiro1` da profundidade da função corrente.

- `store_arg <inteiro1> <inteiro2>`
  Desempilha o valor no topo da pilha de avaliação e atribui-o ao argumento `inteiro2` da função cuja profundidade dista `inteiro1` da profundidade da função corrente.

4.5 Instruções para chamada de funções

As instruções deste grupo processam a chamada de funções e o início e o fim da sua execução, através da manipulação dos registos de ativação:

- `set_arg <inteiro>`
  Esta instrução desempilha da pilha de avaliação o valor para argumento `<inteiro>` da função que vai ser chamada, e põe-o numa localização em que esta lhe poderá aceder.

- `call <inteiro> <etiqueta>`
  Esta instrução efectua a chamada da função cujo nome é dado por `<etiqueta>`, provocando a transferência da execução para a primeira instrução dessa função. Antes de poder ser utilizada, a instrução `set_arg` deverá ser utilizada tantas vezes quantos os argumentos da função, para que estes estejam prontos quando a chamada ocorre.

  O 1º argumento da instrução `call` é a distância entre a profundidade da função chamadora e a da função chamada. Seja f a função em cujo corpo é chamada a função g: se g está definida nas declarações de f, a distância será -1; se g e f estão definidas dentro da mesma função, a distância será 0; se f está definida nas declarações de g, a distância será 1 e assim sucessivamente.

- `locals <inteiro1> <inteiro2>`
  Deve ser a primeira instrução do corpo de uma função e serve para indicar quantos argumentos a função tem, `<inteiro1>`, e quantas variáveis locais declara, `<inteiro2>`.

- `return`
  Esta instrução termina a execução do corpo da função, provocando o retomar da execução da função que a chamou. Se a função devolver algum valor, este é devolvido através da pilha de avaliação, e deverá ter sido colocado antes da instrução `return` ser executada.

4.6 Instruções de salto

Existem três instruções de salto: uma incondicional e duas condicionais:

- `jump <etiqueta>`
  A instrução identificada por `<etiqueta>` é a instrução seguinte a ser executada. A instrução `jump` não tem qualquer outro efeito sobre o estado da máquina.

- `jeq <etiqueta>`
  A instrução seguinte a ser executada é identificada por `<etiqueta>` se os valores nas duas posições no topo da pilha de avaliação forem iguais; caso contrário, será a instrução seguinte à instrução `jeq` na ordem do programa. Em ambos os casos, os dois valores no topo da pilha de avaliação são desempilhados.

- `jlt <etiqueta>`
  Sejam A o valor no topo da pilha de avaliação e B o valor na posição imediatamente abaixo. Se B for menor que A, a próxima instrução a executar é a identificada por `<etiqueta>`; senão, é a que se segue a `jlt` na ordem do programa. Em ambos os casos, os dois valores no topo da pilha de avaliação são desempilhados.

4.7 Instruções de saída

Através das instruções deste grupo é possível, a um programa, enviar informação para o écrã:

- `print`
  Desempilha o valor no topo da pilha de avaliação e escreve-o, como inteiro, no écrã.
- `print_str <string>`
  Escreve a `<string>` no écrã.
- `print_nl`
  Termina a linha corrente na saída do programa. O resultado da execução da próxima instrução de saída aparecerá na linha seguinte do écrã.

5 Implementação

A implementação do leitor de programas TISC deve ler o programa da sua entrada padrão e enviar para a saída padrão da máquina (`System.out`) uma imagem da memória de instruções.

A memória de instruções deverá ter a estrutura considerada mais adequada.

Não faz parte do trabalho fazer um compilador de iPascal para código TISC.

5.1 Linguagem de programação

A leitura de programas TISC deverá ser feita através da programação na linguagem Java, Python ou C (ou outra linguagem que considerem adequada com a aprovação do docente), recorrendo ao gerador de analisadores lexicais `JLex` e ao gerador de analisadores sintáticos `CUP` para a linguagem Java (ou os correspondentes para as linguagens Python ou C).

O ficheiro `ficheiros.zip` contém:

- `regact.lex`: definição do analisador lexical
- `regact.cup`: definição do analisador sintático
- `Main.java`: ficheiro que conterá o programa principal (método `main`)
- `TISC.java`: possível nome para a classe que representará a máquina TISC
- `Makefile`

O conteúdo de qualquer destes ficheiros pode/deve ser alterado, exceto as definições dos *tokens* e a gramática, a não ser para a introdução de ações semânticas.

Os conteúdos dos ficheiros `.java` constituem uma sugestão de estrutura da implementação, podendo ser substituídos por outros, desde que o método `main` pertença à classe `Main` contida no ficheiro `Main.java`. Para as linguagens Python ou C, devem ser feitas as alterações necessárias mantendo os nomes de ficheiros e estrutura sugeridos.

6 Exemplos

O ficheiro `exemplos.zip` contém exemplos de programas em iPascal e uma possível tradução para código TISC:

- `dia.ip`, `dia.tisc`: calcula o ordinal no ano do dia correspondente a uma data
- `factorial.ip`, `factorial.tisc`: calcula iterativamente o factorial de 10
- `factorial_rec.ip`, `factorial_rec.tisc`: versão recursiva do programa anterior
- `fibonacci.ip`, `fibonacci.tisc`: calcula os números de Fibonacci
- `funcfunc.ip`, `funcfunc.tisc`: uma aplicação a uma aplicação
- `mdc.ip`, `mdc.tisc`: calcula do maior divisor comum a dois inteiros
- `mixtura.ip`, `mixtura.tisc`: exemplo do uso de variáveis e funções locais
- `sethi.ip`, `sethi.tisc`: outro exemplo do uso de variáveis e funções locais

7 Relatório

O relatório a entregar deve incluir a descrição das estruturas de dados utilizadas para implementar o leitor de programas TISC.

8 Entrega e Avaliação

O trabalho será realizado por grupos de **dois alunos**. Os elementos a entregar são:

- os ficheiros com o código fonte do trabalho (incluindo os ficheiros com as definições dos analisadores lexical e sintático)
- o Makefile com os comandos para a construção do leitor de programas
- o relatório

Estes elementos deverão ser entregues através do *moodle* num ficheiro de nome `xxx_xxx.zip` (ou `xxx_xxx.tar.gz`), onde `xxx` é o número de cada um dos alunos que fazem parte do grupo. A data de entrega estará disponível no *moodle*.

O trabalho realizado por cada grupo será apresentado à docente em data a combinar. Apesar do trabalho ser de grupo, cada aluno, a título individual, tem a responsabilidade de responder por todo o trabalho. Assim, é indispensável que cada membro do grupo programe efectivamente.

O trabalho deverá ser realizado somente por quem o entrega. A deteção de qualquer tentativa de subverter este princípio terá como consequência, para todos os envolvidos, a impossibilidade de obter aprovação à disciplina este ano lectivo.

A docente da cadeira agradece que qualquer erro detetado no enunciado do trabalho lhe seja comunicado.

Bom trabalho!