INSTITUTO DE INFORMÁTICA UFG

SOFTWARE BÁSICO
TRABALHO EM GRUPO
2019/2

UFG

O trabalho se baseia na implementação de um tradutor de uma linguagem simples, descrita na seção 2, para Assembly.

1. Regras do Trabalho

- O trabalho deve ser feito em grupo de 2 alunos.
- Qualquer plágio (total ou parcial) implicará em nota zero para todos os envolvidos.
- O professor poderá solicitar que o grupo apresente pessoalmente o trabalho, para fins de esclarecimento. Caso o grupo não apresente, o trabalho terá nota zero.
- O grupo deve entregar um arquivo ZIP contendo o código fonte e um relatório (em PDF).
  - O relatório deve indicar o que cada membro do grupo fez no trabalho.
  - Se ficar claro que apenas um membro desenvolveu a maioria ou todo o trabalho, apenas esse membro será pontuado. O outro membro terá nota zero.
- A data de entrega do trabalho será no dia 08 de dezembro de 2019, 23h55m.

2. Descrição da Linguagem

A linguagem é baseada na definição de funções. A definição das funções iniciam com a string “function” seguido de zero a três parâmetros. Os parâmetros podem ser um valor inteiro ou um ponteiro para um array. Os parâmetros inteiros são formados pelo prefixo “pi” (parâmetro inteiro) seguido de um número que serve como um índice que indica se é o primeiro (1), segundo (2) ou terceiro (3) parâmetro. Os parâmetros array são formados pelo prefixo ‘pa’ seguido de um número.

Exemplo 1:

```
function
def
enddef
return ci0
end
```

Exemplo 2:

```
function pi1
def
enddef
return ci0
end
```

Exemplo 3:

```
function pa1, pi2
def
enddef
return ci0
end
```

As funções podem ter variáveis locais (no máximo de 5 variáveis), as quais podem ser variáveis inteiras (*int*) ou arrays de inteiros (*int*). A definição das variáveis são feitas dentro de um bloco “def … enddef”. Cada variável é definida em uma linha em separada. A definição das variáveis inteiras iniciam com “var” e o nome da função têm como prefixo “vi” seguido de índice de identificação. A definição dos arrays iniciam com a string “vet”, o nome dos arrays têm como

prefixo “va” seguido de índice de identificação. Os arrays também têm um tamanho para a quantidade de elementos.

Não haverá duas variáveis com o mesmo índice dentro de uma mesma função, não importando que as variáveis sejam arrays ou variáveis inteiras. Os índices serão incrementais iniciando de 1 para cada função.

A linguagem também possuem constantes inteiras (positivas ou negativas). As constantes começam com o prefixo “ci” seguido com o valor da constante. Por exemplo, o valor “10” deve ser escrito na linguagem como “ci10” e “-123” é escrito como “ci-123”.

Exemplo 1:

```
function
def
var vi1
var vi2
vet va3 size ci10
enddef
return ci0
end
```

Exemplo 2:

```
function pi1
def
var vi1
vet va2 size ci30
enddef
return ci0
end
```

Exemplo 3:

```
function pa1, pi2
def
vet va1 size ci10
vet va2 size ci20
var vi3
enddef
return ci0
end
```

O corpo das funções são um conjunto de comandos. Um comando pode ser uma atribuição de variável inteira, alteração de uma posição do array, recuperação de uma posição de um array e um condicional “*if*” ou um returno de um valor.

Uma atribuição de variável inteira pode ser uma simples atribuição, uma expressão ou o retorno de uma chamada de função. Uma atribuição simples pode ser uma variável inteira recebendo o valor de outra variável inteira ou de uma constante. Uma expressão pode ser as operações de soma, subtração ou multiplicação, e os operandos podem ser variáveis inteiras ou constantes. Por fim, uma variável inteira pode receber o retorno de uma chamada de função.

As chamadas de função são feitas utilizando a palavra-chave “call” seguido do número da função (onde a primeira função definida é a função “1”, a segunda “2”, e assim por diante). Depois do número da função, são passados os parâmetros da (até três) da função chamada. Se a função recebe um parâmetro inteiro, pode-se passar o valor de uma variável inteira ou uma constante, se o parâmetro for um array, deve-se passar um array como referência (ponteiro).

A recuperação de um valor de um array para uma variável inteira utiliza o comando “*get*” e é informado um índice constante. A alteração de uma posição do array é feita utilizando o comando “*set*”, informando o array e o índice (constante) e um valor inteiro (variável inteira ou constante).

O condicional "*if*" possui um único valor de teste que pode ser uma variável ou uma constante. Ela segue a mesma lógica de C, onde zero (0) é false e qualquer valor não zero (positivo ou negativo) é verdadeiro. O corpo possui apenas um único comando que pode ser uma atribuição, acesso a array ou um returno.

Toda função terá ao menos um "return" no final da função. Lembrando que um condicional "*if*" também pode conter um comando "return".

Exemplo 1 (Obs.: os comentários ao lado não existem na linguagem, é só para esclarecimento):

```
function
def
var vi1
var vi2
enddef
vi1 = ci1                              # vi1 = 1
vi2 = vi1                              # vi2 = vi1
vi1 = ci20 + vi2                       # vi1 = 20 + vi2
vi2 = vi1  * ci-5                      # vi2 = vi1 * -5
return vi1                             # return vi1
end

function pi1, pa2
def
var vi1
vet va2 size ci10
enddef
vi1 = pi1 + ci1                        # vi1 = pi1 + 1
set va2 index ci0 with ci2             # va2[0] = 2
set pa2 index ci3 with vi1             # pa1[3] = vi1
get va2 index ci3 to vi1               # vi1 = va2[3]
vi1 = call 1                           # vi1 = f1()
return ci-1                            # return -1
end

function
def
var vi1
var vi2
vet va3 size ci100
enddef
vi1 = ci0                              # vi1 = 0
vi2 = call 2 ci10 va3                  # vi2 = f2(10, &va3)
if vi2                                 # if (vi2 != 0) → vi2 = 1
vi1 = ci1
endif
if vi1                                 # if (vi1 != 0) → return 1
return ci1
endfi
return ci0                             # return 0
end
```

3. Tradução da Linguagem

O programa C do trabalho deve ler as linhas de um arquivo contendo um programa na linguagem simples e imprimir a tradução desse programa em Assembly na tela.

A tradução deve seguir as regras da plataforma x86_64, no sistema Linux: as variáveis locais devem ser alocadas obrigatoriamente na pilha (incluindo os arrays), as variáveis e a pilha devem estar alinhadas, deve-se seguir as regras de passagem de parâmetros e salvamento de registradores (*caller-saved* e *callee-saved*).

4. BNF da Linguagem

Abaixo é descrita uma formalização da linguagem:

```
<prog>     → <func>
           | <func> <prog>
<func>     → <header> <defs> <cmds> 'end\n'
<header>   → 'function' <param> '\n'
<defs>     → 'def\n' <vardef> 'enddef\n'
<vardef>   → 'var' <varint> '\n'
           | 'vet' <varaddr> 'size' <const> '\n'
<cmds>     → <cmd> '\n'
           | <cmd> '\n' <cmds>
<cmd>      → <attr>
           | <arrayget>
           | <arrayset>
           | <ret>
           | <if>
<attr>     → <varaddr> '=' <varaddr>
           | <varint>  '=' <expr>
<expr>     → <valint>
           | <oper>
           | <call>
<oper>     → <valint> <op> <valint>
<call>     → 'call' <num> <valvpc>
<ret>      → 'return' <valint>
<if>       → 'if' <valint> '\n' <body> '\n' 'endif'
<body>     → <attr>
           | <arrayget>
           | <arrayset>
           | <ret>
<op>       → '+' | '-' | '*'
<arrayget> → 'get' <varaddr> 'index' <valint> 'to' <varint>
<arrayset> → 'set' <varaddr> 'index' <valint> 'with' <valint>
<var>      → <varint>
           | <varaddr>
<varint>   → 'vi'<num>
<varaddr>  → 'va'<num>
<param>    → <parint>
           | <paraddr>
<parint>   → 'pi'<num>
<paraddr>  → 'pa'<num>
<const>    → 'ci'<snum>
<valvpc>   → <var>
           | <param>
           | <const>
<valint>   → <varint>
           | <parint>
           | <const>
<num>      → <digit>
           | <digit> <num>
<snum>     → <num>
           | '-'<num>
<digit>    → '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'| '8' | '9'
```