INSTITUTO DE INFORMÁTICA UFG

SOFTWARE BÁSICO
TRABALHO EM GRUPO
2023/1

UFG

O trabalho se baseia na implementação de um tradutor de uma linguagem simples (chamada de *BPL – Bruno's Programming Language*) para Assembly.

1. Regras do Trabalho

- A data de entrega do trabalho será no dia 06 de agosto de 2023, 23h59**.**
- O trabalho deve ser feito em grupo de 3 alunos.
- A tradução deve seguir as regras da plataforma AMD64 no sistema Linux, como estudado na disciplina:
  - Regras de tradução de funções, condicionais, operações aritméticas e arrays.
  - As variáveis locais devem ser alocadas obrigatoriamente na pilha (incluindo os arrays).
  - As variáveis locais e a pilha devem estar alinhadas.
  - Deve-se usar as variáveis locais em registradores designados e só serão salvos na pilha durante a chamada de função.
  - Deve-se usar os parâmetros como os registradores designados e só serão salvos na pilha durante a chamada de função.
  - Deve-se seguir as regras de passagem de parâmetros e retorno.
  - Deve-se seguir as regras de salvamento de registradores (*caller-saved* e *callee-saved*).
- O tradutor deve ser escrito na linguagem C e será compilado e testado no ambiente Linux.
  - Não é permitido o uso de bibliotecas de terceiros para reconhecimento de padrões ou gramáticas.
- Os grupos deverão apresentar o trabalho para o professor, em horários agendados. Caso o grupo não apresente, o trabalho terá nota zero.
- Qualquer plágio (total ou parcial) implicará em nota zero para todos os envolvidos.
- O tradutor deve ler um arquivo em BPL da entrada padrão (e.g., usando o *scanf*) e imprimir a tradução desse programa em Assembly na saída padrão (e.g., usando o *printf*). Pode-se criar um arquivo com a linguagem e utilizar o redirecionamento para testar:

```
$ ./tradutor < prog.blp
```

- O tradutor deve imprimir as posições dos elementos na pilha em forma de comentários, antes da alocação da pilha. Isso inclui variáveis locais ou qualquer posição de salvamento de registradores (como se fosse o desenho da pilha). Por exemplo:

```
# vi1: -4
# va2: -16
# vi3: -32
# rbx: -36
subq $48, %rsp
```

- Caso omissos devem ser tratados com o professor. Não tente assumir qualquer coisa se estiver com dúvida.

2. Descrição da Linguagem

A linguagem é baseada na definição de funções (uma ou várias). As funções sempre retornam algum valor.

2.1. Definição de Função

A definição de função inicia com a palavra-chave **`function`**, seguido pelo nome da função e de zero a três parâmetros. O nome da função segue o padrão **`fN`**, onde ***`N`*** é um índice único começando de 1 (1, 2, 3, etc.).

Os parâmetros podem ser um valor inteiro (*int*) ou um ponteiro para um array de inteiro (*int**). O nome do parâmetro inteiro segue o padrão **`piN`**, onde ***`N`*** é um índice que indica se é o primeiro (1), segundo (2) ou terceiro (3) parâmetro. O nome do parâmetro array inteiro segue o padrão **`paN`**, onde ***`N`*** é um índice que indica se é o primeiro (1), segundo (2) ou terceiro (3) parâmetro.

Na tradução, os parâmetros devem ser mantidos e referenciados pelos registradores %rdi, %rsi e %rdx. Ao chamar uma função, esses registradores devem ser salvos na pilha e devem ser recuperados tão logo a chamada termine. Deve-se salvar apenas os parâmetros definidos pela função, por exemplo, se a função definir um parâmetro, apenas %rdi deve ser salvo. Mas, se a função definir 3 parâmetros, os registradores %rdi, %rsi e %rdx devem ser salvos.

Assuma que os índices dos parâmetros sempre estarão corretos.

Exemplo:

```
function f1
def
enddef
return ci0
end

function f2 pi1
def
enddef
return ci0
end

function f3 pa1 pi2 pa3
def
enddef
return ci0
end
```

2.2. Variáveis Locais

Uma função pode ter no máximo quatro variáveis inteiras de pilha (*int*), quatro variáveis inteiras de registrador (*int*) ou quatro arrays de inteiros (*int*[]) - no máximo, doze variáveis locais. Tanto as variáveis inteiras de pilha ou os arrays inteiros devem ser alocados na pilha e não possuem valor inicial (têm “lixo” de memória).

No caso de variáveis locais de registrador, deve-se alocar um registrador para cada variável. Esses registradores devem ser efetivamente utilizado no corpo da função.

As variáveis são definidas uma por linha, dentro de um bloco **`def-enddef:`**

```
def
  ...
enddef
```

O bloco de definição de variável é obrigatório, mas pode ser vazio se não houver variáveis locais a serem definidas.

A definição das variáveis inteiras de pilha iniciam com a palavra-chave **`var`**, seguida do nome da variável. O nome segue o padrão **`viN`**, onde ***`N`*** é um índice de identificação, por exemplo, **`vi1`**, **`vi2`** ou **`vi3`**.

A definição dos arrays inteiros iniciam com a palavra-chave **vet**, seguida do nome do array, seguido da palavra-chave **size** e então uma constante inteira informando o tamanho do array. O nome do array segue o padrão **va*N*** (por exemplo, **va3** ou **va5**).

Uma constante inteira tem o formato **ci*v***, onde ***v*** é o valor da constante, por exemplo, **ci5** (5), **ci-15** (-15), **ci1024** (1024), **ci-8273** (-8273), etc. No caso do tamanho dos arrays, sempre deve ser uma constante positiva não nula (> 0).

A definição de variável de registrador iniciam com a palavra-chave **reg**, seguida do nome da variável. O nome segue o padrão **ri*N***, onde ***N*** é um índice de identificação, por exemplo, **ri1**, **ri4** ou **ri6**.

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

Exemplo:

```
function f1
def
enddef
return ci0
end

function f2 pi1
def
var vi1
vet va2 size ci30
var vi3
enddef
return ci-1
end

function f3 pa1 pi2
def
vet va1 size ci10
vet va2 size ci20
var vi3
reg ri4
var vi5
reg ri6
enddef
return ci5
end
```

2.3. Corpo da Função

O corpo da função é um conjunto de comandos que inicia depois da definição das variáveis. Um comando pode ser (i) atribuição de variável inteira, (ii) alteração de uma posição do array, (iii) recuperação de valor de uma posição de um array, (iv) condicional **if** ou (v) returno de um valor.

2.3.1. Atribuição

Uma atribuição de variável inteira pode ser uma atribuição simples, uma expressão ou o retorno de uma chamada de função. Uma atribuição simples pode ser uma variável inteira recebendo o valor de outra variável, um parâmetro inteiro ou uma constante. Uma expressão pode ser as operações de soma, subtração, multiplicação ou divisão, sendo que os operandos podem ser variáveis inteiras, parâmetros inteiros ou constantes (não pode ter chamada de função ou arrays na expressão). Por fim, uma variável inteira pode receber o retorno de uma chamada de função.

Exemplo:

```
function f1 pi1
def
var vi1
reg ri2
enddef
vi1 = ci1                           # vi1 = 1
ri2 = vi1                           # ri2 = vi1
vi1 = pi1 + ri2                     # vi1 = pi1 + ri2
ri2 = vi1 * ci-5                    # ri2 = vi1 * -5
return vi1
end
```

Obs: A linguagem não tem comentários, eles foram colocados no exemplo como forma de esclarecimento.

2.3.2. Chamada de Função

As chamadas de função são permitidas na atribuição e utiliza a palavra-chave **`call`** seguida do nome da função. Depois do nome da função são passados os parâmetros para função a ser chamada (até três parâmetros). Se a função recebe um parâmetro inteiro, pode-se passar o valor de uma variável inteira (pilha ou registrador), um parâmetro inteiro ou uma constante inteira. Se o parâmetro for um ponteiro para array, pode-se passar um array local (o tradutor deve obter o ponteiro do array local e passar para a função) ou um parâmetro array (que já é um ponteiro, passado por outra função).

Exemplo:

```
function f1 pi1 pa2
def
enddef
return pi1
end

function f2 pa1
def
reg ri1
var vi2
vet va3 size ci30
enddef
ri1 = ci1
vi2 = call f1 ri1 va3               # vi2 = f1(ri1, &va3)
vi2 = call f1 ci5 pa1               # vi2 = f1(5, pa1)
return vi2
end
```

Obs: A linguagem não tem comentários, eles foram colocados no exemplo como forma de esclarecimento.

2.3.3. Acesso ao Array

A recuperação de um valor de um array utiliza o comando **`get`**, no seguinte formato:

```
get array index índice to destino
```

Onde:

- *array:* um array local ou um parâmetro array.
- *índice:* uma constante inteira não-negativa (índice do vetor – inicia em 0).
- *destino:* uma variável local inteira ou um parâmetro inteiro.

Para modificar uma posição de um array, utiliza-se o comando **`set`**:

```
set array index índice with valor
```

Onde:

- *array:* um array local ou um parâmetro array.
- *índice:* uma constante inteira não-negativa (índice do vetor – inicia em 0).
- *valor:* uma variável local inteira (pilha ou registrador), um parâmetro inteiro ou uma constante inteira.

Exemplo:

```
function f1 pi1 pa2
def
var vi1
vet va2 size ci10
reg ri3
enddef
vi1 = pi1 + ci1
set va2 index ci5 with ci2          # va2[5] = 2
set pa2 index ci0 with vi1          # pa2[0] = vi1
get va2 index ci8 to vi1            # vi1 = va2[8]
get pa2 index ci4 to ri3            # ri3 = pa2[4]
return pi1
end
```

Obs: A linguagem não tem comentários, eles foram colocados no exemplo como forma de esclarecimento.

2.3.4. Condicional

O condicional **`if`** não possui *else*. A expressão só pode comparar variáveis inteiras (pilha ou registrador), parâmetros inteiros ou constantes inteiras. Os operadores relacionais são:

- `eq`: igual
- `ne`: não igual
- `lt`: menor
- `le`: menor igual
- `gt`: maior
- `ge`: maior igual

O formato do **`if`** é:

```
if condição
comando
endif
```

Onde:

- *condição*: comparação entre variável inteira e/ou parâmetro inteiro.
- *comando*: atribuição, acesso a array (*get/set*) ou retorno.

Exemplo:

```
function f1 pi1
def
var vi1
vet va2 size ci10
var vi3
enddef
vi1 = ci3
vi3 = ci4
if vi1 ne vi3              # if (vi1 != vi3) → vi1 = 0
vi1 = ci0
endif
if vi1 lt vi3              # if (vi1 < vi3) → vi1 = va2[8]
get va2 index ci8 to vi1
endif
if vi3 le vi1              # if (vi3 <= vi1) → return vi1
return vi1
endif
if pi1 le ci10             # if (pi1 < 10) → return 0
return ci0
endif
if ci-10 gt ci10           # if (-10 > 10) → vi3 = va2[2]
get va2 index ci2 to vi3
endif
return ci-1
end
```

Obs: A linguagem não tem comentários, eles foram colocados no exemplo como forma de esclarecimento.

2.3.5. Retorno da Função

O comando **`return`** só poderá aparecer como último comando do corpo da função ou no corpo do **`if`**. Toda função terá obrigatoriamente um **`return`** como último comando do corpo. O formato do comando é:

`return` *valor*

Onde:

- *valor:* uma variável inteira, um parâmetro inteiro ou uma constante inteira.

3. BNF da Linguagem

```
<prog>     → <func>
           | <func> <prog>

<func>     → <header> <defs> <cmds> <ret> '\n' 'end' '\n'

<header>   → 'function' <fname> <params> '\n'
<fname>    → 'f'<num>
<params>   → ε
           | <param> <params>
<param>    → <parint>
           | <pararr>
<parint>   → 'pi'<num>
<pararr>   → 'pa'<num>

<defs>     → 'def' '\n' <vardef> 'enddef' '\n'
<vardef>   → 'var' <varint> '\n'
           | 'vet' <vararr> 'size' <const> '\n'
<varint>   → 'vi'<num>
           | 'ri'<num>
<vararr>   → 'va'<num>
<const>    → 'ci'<snum>

<cmds>     → <cmd> '\n'
           | <cmd> '\n' <cmds>
<cmd>      → <attr>
           | <arrayget>
           | <arrayset>
           | <if>

<attr>     → <varint>  '=' <expr>
<expr>     → <valint>
           | <oper>
           | <call>

<valint>   → <varint>
           | <parint>
           | <const>

<oper>     → <valint> <op> <valint>
<op>       → '+' | '-' | '*' | '/'

<call>     → 'call' <fname> <args>

<args>     → ε
           | <arg> <args>
<arg>      → <valint>
           | <array>
<array>    → <vararr>
           | <pararr>
```

```
<arrayget> → 'get' <array> 'index' <const> 'to'   <varint>
<arrayset> → 'set' <array> 'index' <const> 'with' <valint>

<if> → 'if' <valint> <oprel> <valint> '\n' <body> '\n' 'endif'

<oprel> → 'eq' | 'ne' | 'lt' | 'le' | 'gt' | 'ge'

<body> → <attr>
       | <arrayget>
       | <arrayset>
       | <ret>

<ret>  → 'return' <valint>

<num>  → <digit>
       | <digit> <num>

<snum> → <num>
       | '-'<num>

<digit> → '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'| '8' | '9'
```