Universidade da Beira Interior

Departamento de Informática

Compilador em OCaml da linguagem Arith para Assembly MIPS

Projeto realizado por:

André Martins Nº37413

João Brito Nº37880

Processamento de Linguagens

Professor Simão Melo de Sousa

11 de janeiro de 2019

Objetivos

O objetivo deste projeto passa pelo desenvolvimento de um compilador preparado para receber programas escritos na linguagem arith. Estes contêm várias instruções (mais se acrescenta adiante) que, antes de executadas, são passadas para o seu equivalente em Assembly MIPS.

Para tal, foi utilizado o compilador previamente preparado na aula prática 1, onde a arquitetura alvo era Assembly x86_64 e as instruções suportadas limitavam-se a impressão no *stdout* e operações aritméticas. Também foi utilizado um ficheiro disponibilizado com todos os pormenores implementados relativamente aos registos e instruções MIPS.

Motivação

Este projeto tem várias aplicações reais, porque representa bem como é desenvolvido um compilador para uma determinada linguagem. Proporcionou uma melhor compreensão do funcionamento de linguagens de baixo nível, como é o caso do Assembly.

Descrição

Funcionalidades

Este compilador permite ao utilizador desenvolver um programa construído com o seguinte:

- Declaração de variáveis (“set”);
- Atribuição de valores a variáveis;
- Adições, subtrações, multiplicações e divisões;
- Condições (“if then else”);
- Impressão no *stdout* (“print”);
- Ciclos (“while do”);
- Comentários.

Após o desenvolvimento desse programa, o compilador trata de gerar um ficheiro com extensão .s, cujo conteúdo é o equivalente ao ficheiro de alto nível (“test.exp”) em MIPS. Para compilar, basta fazer uso do comando “make”.

Estrutura do Compilador

Para este compilador foram desenvolvidos e adaptados diversos ficheiros do compilador original, sendo eles:

- **Árvore de sintaxe abstrata** (ast.mli)

Neste ficheiro está descrita a sintaxe abstrata da linguagem Arith, nomeadamente:

Expressões

expr ::=

| I (inteiros)

| Var (variáveis)

| Op (operações aritméticas)

Op ::=

| Add (adição)

| Sub (subtração)

| Mul (multiplicação)

| Div (divisão)

Testes

test ::=

| B (booleanos)

| Comp (==)

| Big (>)

| BigEqual (>=)

| Small (<)

| SmallEqual (<=)

| Neg (!)

Instruções

inst ::=

| Dec (declaração)

| Set (atribuição)

| Print (impressão)

| Cond (condicional)

| Loop (ciclo)

| Nop (não fazer nada)

- **Parser** (parser.mly)

Este ficheiro conjuga os *tokens* recebidos em regras pré-definidas. Considera-se então a seguinte sintaxe:

“set x = expr;” (declaração)

“x = expr;” (atribuição)

“print expr;” (impressão)

“# texto” (comentário)

“if test then (inst*) else (inst*)” (condicional)

“while test do (inst*)” (ciclo)

- **Lexer** (lexer.mll)

Este ficheiro identifica os *tokens* do texto-limpo, agrupados no parser.

- **Mips** (mips.ml e mips.mli)

Estas bibliotecas contêm mecanismos e declarações que permitem a escrita no ficheiro .s final.

- **Compile** (compile.ml)

Configura o ficheiro mais alterado, no qual foram acrescentados mecanismos de inserção de código MIPS no ficheiro .s final.

- **Makefile**

Este ficheiro é responsável pelo processo de compilação.s

Implementações

Nas imagens seguintes serão comparadas as instruções na linguagem Arith e as suas equivalentes em Assembly MIPS.

NOTA: Todas as instruções básicas em Arith são terminadas com “;”.

A declaração faz-se com a palavra reservada “set”:

```
set x = 3; # declaracao de variaveis
```

```
.data
newline:    .asciiz "\n"
x:  .word 3
```

A atribuição só é possível após uma declaração, caso contrário resulta em erro de compilação:

```
x = 4; # atribuicao de valores a variaveis
```

```
li $t0, 4
sub $sp, $sp, 4
sw $t0, ($sp)
lw $t0, ($sp)
add $sp, $sp, 4
sw $t0, x
```

A impressão é efetuada com a palavra reservada “print”:

```
print x; # impressao no stdout
```

```
lw $t0, x
sub $sp, $sp, 4
sw $t0, ($sp)
lw $t0, ($sp)
add $sp, $sp, 4
li $v0, 1
move $a0, $t0
syscall
```

A construção “if-else” faz uso, no mínimo, das palavras reservadas “if”, “then” e “then” e os ramos/blocos de instruções são delimitados por “(” e “)”. É possível, ainda, aninhar estas construções:

```
if x>4 then (print x;) else (pass;) # construcao if-else
```

```
if true then (if true then (print 3;) else (pass;)) else (pass;) # construcoes if-else aninhadas
```

```
lw $t0, x
sub $sp, $sp, 4
sw $t0, ($sp)
li $t0, 4
sub $sp, $sp, 4
sw $t0, ($sp)
lw $t0, ($sp)
add $sp, $sp, 4
lw $t1, ($sp)
add $sp, $sp, 4
ble $t1, $t0, else1
lw $t0, x
sub $sp, $sp, 4
sw $t0, ($sp)
lw $t0, ($sp)
add $sp, $sp, 4
li $v0, 1
move $a0, $t0
syscall
li $v0, 4
la $a0, newline
syscall
j main1
```

```
else1:
main1:
```

O ciclo “while” faz uso, no mínimo, das palavras reservadas “while” e “do”. Partilha da estrutura sintática da construção condicional:

```
while x<6 do (x = x + 1;) # ciclo while
```

```
loop1:
    lw $t0, x
    sub $sp, $sp, 4
    sw $t0, ($sp)
    li $t0, 6
    sub $sp, $sp, 4
    sw $t0, ($sp)
    lw $t0, ($sp)
    add $sp, $sp, 4
    lw $t1, ($sp)
    add $sp, $sp, 4
    bge $t1, $t0, exit1
    lw $t0, x
    sub $sp, $sp, 4
    sw $t0, ($sp)
    li $t0, 1
    sub $sp, $sp, 4
    sw $t0, ($sp)
    lw $t0, ($sp)
    add $sp, $sp, 4
    lw $t1, ($sp)
    add $sp, $sp, 4
    add $t0, $t0, $t1
    sub $sp, $sp, 4
    sw $t0, ($sp)
    lw $t0, ($sp)
    add $sp, $sp, 4
    sw $t0, x
    j loop1
exit1:
```

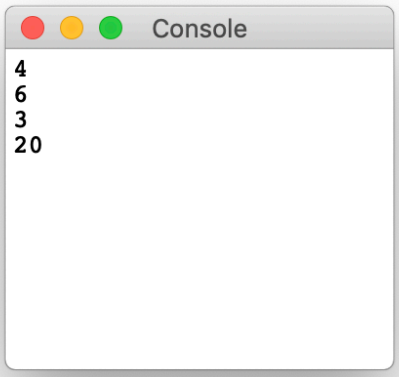
Avaliação do programa

Relativamente à avaliação do programa, devem ser efetuados vários testes, sendo que a maioria desses testes devem ser feitos no ficheiro de teste (“test.exp”). Todos eles passam pela elaboração de um programa baseado na linguagem anteriormente descrita. A implementação desses programas pode ser feita da maneira que o programador bem entender, desde que a sequência de ações seja lógica e esteja de acordo com o que o compilador espera.

Resultados Experimentais

De seguida será mostrado um ficheiro de teste que sumariza as capacidades da linguagem desenvolvida e o respetivo *output*:

```
# ficheiro de teste para Arithc 2.0

set x = 3; # declaracao de variaveis

x = 4; # atribuicao de valores a variaveis

print x; # impressao no stdout

x = 2 + 2; # operacoes de soma, subtracao, multiplicacao e divisao

if x>4 then (print x;) else (pass;) # construcao if-else

while x<6 do (x = x + 1;) # ciclo while

print x; # deve imprimir o valor 6

if true then (if true then (print 3;) else (pass;)) else (pass;) # construcoes if-else aninhadas

if !3<3 then (print 20;) else (pass;) # deve imprimir 20
```



Melhoramentos

Este compilador pode ser melhorado através da implementação de outros tipos de dados primitivos, funções ou até tipos definidos pelo programador. Contudo, o projeto atual configura uma base sólida passível de ser melhorada.

Conclusão

Este projeto partiu da adaptação de um compilador para Arith usando Assembly x86_64. Efetuada a adaptação para MIPS e uma amplificação considerável para ter mais funcionalidades, o resultado final providenciou uma boa perspetiva do processo de compilação de uma linguagem.