Tópicos Avançados de Compilação
Relatório do Segundo Trabalho Prático

João Calhau, m36764

January 17, 2017

UNIVERSIDADE
DE ÉVORA

1 Introdução

O código entrege diz respeito á parte que vale 9/10 (Instruction Selection), não fiz a 1/10 (Simple register allocation) devido ás entregas de trabalho e exames das outras cadeiras.
O código já contem comentários acerca do funcionamento do mesmo.
Este relatório serve apenas para demonstrar as escolhas que fiz acerca da tradução do código IR para MIPS.

2 Selecção de instruções

2.1 Prologue

```
sw $fp, -4($sp)                          #Guardar o frame pointer da função que chamou
addiu $sp, $fp, -4                       #Guardar o frame pointer da função chamada
sw $ra, -4($fp)                          #Guardar o return adress
addiu $sp, $fp, -(4 + #locals * 4) #Alocar espaço para os locals e return adress
```

2.2 Epilogue

```
lw $ra, -4($fp)                          #Restaurar o return adress
addiu $sp, $fp, (4 + #args * 4) #Restaurar o stack pointer da função que chamou
lw $fp, 0($fp)                           #Restaurar o frame pointer da função que chamou
jr $ra                                   #Retornar da função
```

2.3 Instruções com 2 argumentos

```
t1 <- i_eq t2, t3:
      subu t1, t2, t3
      sltiu t1, t1, 1

t1 <- i_ne:
      subu t1, t2, t3
      sltu t1, $0, t1

t1 <- i_lt t2, t3:
      slt t1, t2, t3

t1 <- i_le t2, t3:
      slt t1, t2, t3
      xori t1, t1, 1

t1 <- i_add t2, t3:
      addu t1, t2 ,t3

t1 <- i_sub t2, t3:
      subu t1 ,t2 ,t3

t1 <- i_mul t2, t3:
      mult t2, t3
      mflo t1

t1 <- i_div t2, t3:
      div t2, t3
      mflo t1

t1 <- mod t2, t3:
      div t2, t3
      mfhi t1
```

2.4 Instruções com 1 argumento

t1 <- not t2:
    xori t1, t2, 1

t1 <- i_inv t2:
    subu t1, $0, t2

t1 <- i_copy t2:
    add t1, t2, $0

2.5 Instruções load

t1 <- i_gload @name:
    la t1, name
    lw t1, 0(t1)

t1 <- i_lload @name:
    lw t1, -X($fp)

t1 <- i_aload @name:
    lw t1, X($fp)

Tenho noção que no meu código não tenho a busca do valor de X correta, visto eu apenas ir buscar o ultimo local sempre. Inicialmente tinha feito de uma maneira que guardava cada local numa struct propria, e depois era só ir a essa struct buscar os valores quando fosse preciso. Tudo estava a funcionar, mas quando mexi no código outra vez, fiz qualquer coisa (que não sei o que foi) e ao correr dava-me sempre segmentation fault. Como já estava um bocado em cima da hora de entrega, decidi voltar a uma versão anterior e ficou assim.

2.6 Instruções store

t1 <- i_gstore @name:
    la $at, name
    sw t1, 0($at)

t1 <- i_lstore @name:
    sw t1, -X($fp)

t1 <- i_astore @name:
    sw t1, X($fp)

O problema das load instructions mantem-se para as store, visto a maneira que tinha feita ser igual ás load.

2.7 Instrução value

t1 <- i_value X: #para numeros menores que 65536
    ori t1, $0, X

t1 <- i_value X: #para numeros maiores que 65536
    lui t1, X_1
    ori t1, t1, X_2

$X_1 = X >> 16$ e $X_2 = X - (X_1 * 65536)$.

2.8 Instrução jump e cjump

jump l0:
    j l$0

cjump t1, l0, l1:
    beq t1, $0, l$1
    j l$0

2.9 Instruções return

i_return t1:
    or $v0, $0, t1

return:
    Não é preciso instrução nenhuma

2.10 Instruções call

t1 <- i_call @name, [t2, t3]:
    addiu$sp, $sp, -4
    sw t2, 0($sp)
    addiu $sp, $sp, -4
    sw t3, 0($sp)
    jal name
    or t1, $0, $v0

call @name, [t1, t2]:
    addiu $sp, $sp, -4
    sw t1, 0($sp)
    addiu $sp, $sp, -4
    sw t2, 0($sp)
    jal name

A unica parte que vai mudar nestas duas instruções é as primeiras 4 linhas de código que podem ser 0 ou mais, consoante a quantidade de argumentos que a função que está a ser chamada leva. Neste exemplo só leva 2 argumentos, por isso há 4 linhas no inicio.

2.11 Instruções print

```
i_print t1:
    i_print$ t1

b_print t1:
    b_print$ t1
```

Visto estar a fazer a parte 9/10 do trabalho não foi preciso fazer o macro inicial para os prints por isso não falarei dele aqui.