Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Tópicos Avançados de Compilação

Geração de Código Para TACL

Trabalho Elaborado Por:
Marlene Oliveira, Nº 11327

23 de Janeiro de 2015

Conteúdo

1 Introdução

O objetivo deste trabalho consiste na criação de um gerador de código que traduz a representação intermédia de um programa TACL para instruções MIPS, produzindo o código *assembly* pretendido. O trabalho tem como principal foco a *instruction selection*. A *instruction selection* consiste na "tradução das instruções da representação intermédia para um conjunto de instruções da máquina de destino". Além disto, o trabalho pode ter também uma componente de alocação de registos. Esta componente pode ser simples (os *live ranges* dos registos temporários não ultrapassam as fronteiras dos *basic blocks*) ou mais complexa (os registos temporários têm *live ranges* que passam as fronteiras dos *basic blocks*).

A implementação do gerador de código descrita neste relatório inclui *instruction selection* e alocação de registos simples. A *instruction selection* é feita instrução a instrução, não sendo usadas janelas deslizantes (*sliding windows*). Neste passo, as instruções da representação intermédia são traduzidas para instruções MIPS. A alocação de registos simples segue o algoritmo de *simple register allocation* descrito nas aulas[1]. A representação intermédia usada é a que se encontra no formato standard em prolog. A linguagem usada para implementar o gerador de código foi Prolog. Todo o código desenvolvido para este trabalho pode ser consultado nos ficheiros: *codegen.pl, codegenarit.pl, codegenLS.pl e codegenregaux.pl*.

2 Alocação de Registos Simples

O algoritmo usado para a alocação de registos é o apresentado nos slides das aulas que dizem respeito à *instruction selection* [1]. O algoritmo referido é o seguinte:

- Seja $r_0, r_1, \ldots, r_{k-1}$ o conjunto dos registos disponíveis e $n = 0$ o número de registos em uso atualmente.
- Para cada instrução i da representação intermédia:
  - Decrementar a n o número de temporários usados, se i usar temporários;
  - Se $n > 0$ e i for uma instrução *call*, gerar código para guardar os registos $r_0, \ldots, r_{n-1}$ antes do *call* e gerar código para restaurar os valores destes registos depois do *call*;
  - Se i cria um novo valor, atribuir o temporário ao registo r_n e $n = n + 1$;
  - Gerar código para i, substituindo os temporários pelos registos que lhes foram atribuídos.

Para auxiliar na aplicação deste algoritmo, foram criados alguns predicados auxiliares. Estes predicados são descritos na secção 3.4.

3 *Instruction Selection*

A tradução das instruções da representação intermédia (IR) para instruções MIPS efetuada está descrita nas secções seguintes.

3.1 Instruções Aritméticas e Comparações

As instruções aritméticas e comparações da representação intermédia para as quais é feita a geração de código são, de acordo com o enunciado, apenas aquelas que lidam com inteiros. Assim, foi feita a tradução das seguintes instruções aritméticas da representação intermédia: *i_add* (soma), *i_sub* (diferença), *i_mul* (multiplicação), *i_div* (divisão), *i_mod* (resto da divisão inteira), *i_inv* (valor simétrico). Sempre que possível, usam-se instruções MIPS *unsigned*. A tradução usada para cada uma das instruções referidas anteriormente pode ser consultada na tabela 3.1.

Instrução da Representação Intermédia	**Instrução MIPS**
i_add t_i, t_j, t_k	addu t_i, t_j, t_k
i_sub t_i, t_j, t_k	subu t_i, t_j, t_k
i_mul t_i, t_j, t_k	mult t_j, t_k mflo t_i
i_div t_i, t_j, t_k	div t_j, t_k mflo t_i
i_mod t_i, t_j, t_k	div t_j, t_k mfhi t_i
i_inv t_i, t_j	subu $t_i, \$0, t_j$

Tabela 1: Instruções da representação intermédia de TACL e correspondentes instruções MIPS para que foram traduzidas.

De modo a que seja possível aplicar o algoritmo de alocação de registos simples descrito na secção 2, em cada uma destas instruções aritméticas são usados predicados que permitem decrementar o número de registos atualmente em uso, obter os registos correspondentes aos argumentos da instrução (aqui representados como t_j e t_k), guardar o novo registo criado para armazenar o valor obtido após a operação e, finalmente, incrementar o número de registos atualmente em uso. Um exemplo de um predicado que gera o código de uma destas instruções pode ser visto de seguida.

```
1 i_add(X,Y,Z):-
2   decrementRegC(2,B),          % Decrementar o num. de registos atuais.
3   tempToReg(Y,C),              % Obter o registo de um dos argumentos.
4   tempToReg(Z,D),              % Obter o registo do outro argumento.
5   format(atom(T),"t~w",[B]),    % Criar um novo registo.
```

```
6   assert(tempToReg(X,T)),        % Guardar o novo registo numa variavel global
        auxiliar.
7   format("\taddu\t$~w, $~w, $~w\n",[T,C,D]), % Gerar o codigo para a instrucao
        i_add.
8   incrementRegC(1,_).            % Incrementar o num. de registos atuais.
```

Em termos de estrutura, as comparações foram implementadas de um modo semelhante ao usado para as instruções aritméticas. A tradução usada para as comparações pode ser consultada na tabela 3.1. A estrutura dos predicados é semelhante à do exemplo apresentado para as instruções aritméticas.

Os predicados que geram o código para estas instruções podem ser consultados no ficheiro *codegenarit.pl*.

Instrução da Representação Intermédia	**Instrução MIPS**
i_lt t_i, t_j, t_k	slt t_i, t_j, t_k
i_eq t_i, t_j, t_k	xor t_i, t_j, t_k sltiu $t_i, t_i, 1$
i_le t_i, t_j, t_k	slt t_i, t_j, t_k xori $t_i, t_i, 1$
i_ne t_i, t_j, t_k	xor t_i, t_j, t_k sltiu $t_i, 0, t_i$

Tabela 2: Instruções da representação intermédia para as comparações e correspondentes instruções MIPS para as quais foram traduzidas.

3.2 Instruções *load* e *store*

À semelhança do que aconteceu com as instruções aritméticas e comparações, também só foram consideradas instruções de *load* e *store* da representação intermédia que lidem com inteiros. Assim, as instruções consideradas foram: *i_load*, *i_aload*, *i_lload*, *i_store*, *i_astore*, *i_lstore*, *i_value*. As traduções usadas para cada uma destas instruções podem ser consultadas na tabela 3.2. Os predicados que geram o código para estas instruções podem ser consultados no ficheiro *codegenLS.pl*.

*D*e modo a que fosse possível aplicar o algoritmo de alocação de registos simples, também foram usados predicados auxiliares nos predicados que geram o código dos *loads* e dos *stores*. Estes predicados permitiram obter o nome da função atual, usar essa informação para consultar o registo de ativação da mesma e obter o *offset* das variáveis locais ou argumentos pretendidos e atualizar o número de registos atualmente em uso. No caso do predicado que permite carregar um inteiro para uma variável, os predicados auxiliares permitem obter o número de registos atualmente em uso, incrementar esse valor e guardar numa variável global auxiliar o novo registo criado. Exemplos de predicados que

Instrução da Representação Intermédia	Instrução MIPS
i_value x	ori $t_i, \$0, x$
i_load t_i, x	la t_i, x lw $t_i, 0(t_i)$
i_aload t_i, x	lw $t_i, offset(\$fp)$
i_lload t_i, x	lw $t_i, offset(\$fp)$
i_store t_i, x	la t_i, x sw $t_i, 0(t_i)$
i_astore t_i, x	sw $t_i, offset(\$fp)$
i_lstore t_i, x	sw $t_i, offset(\$fp)$

Tabela 3: Instruções da representação intermédia para as instruções *load* e *store* e correspondentes instruções MIPS para as quais foram traduzidas.

geram o código de uma instrução *load* e de uma instrução *store* podem ser consultados de seguida.

```
1  i_lload(X,N):-
2     currentF(F),              % Obter o nome da funcao atual.
3     actRec(arg,N,_,FP,F),     % Consultar o registo de ativacao para obter o
          offset.
4     currentReg(A),            % Obter o num. de registos .
5     incrementRegC(1,_),       % Incrementar o num. de registos .
6     format(atom(T),"t~w",[A]),  % Criar um novo registo .
7     assert(tempToReg(X,T)),   % Guardar o registo novo numa variavel global
          auxiliar.
8     format("\tlw\t$~w, ~w($fp)\n",[T,FP]). % Gerar o codigo para a instrucao
          i_aload.
9
10 i_lstore(N,Y):-
11    currentF(F),              % Obter o nome da funcao atual.
12    actRec(local,N,_,FP,F),   % Consultar o registo de ativacao para obter o
          offset.
13    decrementRegC(1,_),       % Decrementar o num. de registos .
14    tempToReg(Y,X),           % Obter o registo correspondente.
15    format("\tsw\t$~w, ~w($fp)\n",[X,FP]). % Gerar o codigo da instrucao
          i_lstore.
```

3.3 Outras Instruções

Além das instruções apresentadas nas secções anteriores, foram também incluídos predicados que geram o código de outras instruções, tais como: *prints*, *jumps*, *calls* e *returns*.

A tradução usada para cada uma das instruções referidas pode ser consultada na tabela 3.3. Os predicados que geram o código para estas instruções podem ser consultados no ficheiro *codegen.pl*.

Instrução da Representação Intermédia	**Instrução MIPS**
i_print t_i	i_print$ t_i
b_print t_i	b_print$ t_i
jump l_i	j l_i
cjump t_i, l_x, l_y	beq t_i, \$0, l_y j l_x

i_call $t_i, X, [t_k]$	addiu \$$sp$, \$$sp$, -4
	sw t_k, 0(\$$sp$)
	jal X

call $X, [t_k]$	addiu \$$sp$, \$$sp$, -4
	sw t_k, 0(\$$sp$)
	jal X

i_return t_i	or v_0, \$0, t_i
b_return t_i	or v_0, \$0, t_i

Tabela 4: Instruções da representação intermédia para as restantes instruções e correspondentes instruções MIPS para as quais foram traduzidas.

Todas as instruções apresentadas utilizam alguns predicados auxiliares para que seja possível aplicar o algoritmo de alocação de registos simples. As instruções *call* utilizam alguns predicados auxiliares adicionais, de modo a que seja possível guardar os registos em uso e, depois de ser gerado o código da instrução, restaurar os valores dos registos guardados anteriormente. As instruções que correspondem aos *prints* usam ainda macros, que são definidas no início de cada função. Existem também predicados que permitem gerar o prólogo e o epílogo dos programas MIPS resultantes da tradução.

3.3.1 Variáveis Globais

A geração de código para as variáveis globais é relativamente simples. De cada vez que existe uma nova variável global, é usado um predicado semelhante ao seguinte:

```
1 id(X,var,int,Y):-
2   write('\t.data\n'),          % Escrever a diretiva .data.
3   format("~w:\t.word ~w\n",[X,Y]). % Gerar o codigo.
```

Caso a variável global não tenha um valor atribuído, o predicado usado é semelhante ao seguinte:

```
1 id(X,var,int):-
2   write('\t.data\n'),          % Escrever a diretiva .data.
```

```
3    format("~w:\t.space 4\n",[X]). % Gerar o codigo.
```

A única coisa que estes predicados fazem é gerar o código que corresponde à declaração da respetiva variável global após ser gerada a diretiva *.data*. Visto que a ordem das secções *.data* não é importante[1], estas podem estar divididas em várias partes. Assim, optou-se por gerar o código respeitante a esta diretiva sempre que se gera o código de uma variável global.

3.4 Termos Globais e Predicados Auxiliares

De modo a auxiliar na geração de código e alocação de registos, foram criadas alguns termos auxiliares que são guardados como variáveis globais do gerador de código. Estas variáveis globais são as seguintes:

- *localAloc(A,B,C)* - este termo permite guardar o número correspondente ao número de bytes alocados para as variáveis locais (aqui representado por A), o número de bytes correspondentes ao espaço alocado para argumentos da função (aqui representado por B) e o nome da função a que estes valores correspondem (representado por C). Este termo é usado antes de ser gerado o código do prólogo e do epílogo do programa MIPS;

- *actRec* - permite guardar a informação relativa ao registo de ativação de cada função. Para distinguir cada função, é usado um parâmetro que corresponde ao nome da função a que corresponde cada elemento do registo. Para distinguir variáveis locais e argumentos existe também um parâmetro neste termo. Consultado este termo é possível obter o *offset* necessário para fazer o *load* ou *store* de cada variável local ou argumento da função;

- *fpOffA* e *fpOffL* - termos que permitem obter os *offsets* atuais dos argumentos e variáveis locais, respetivamente. São usados durante a criação do registo de ativação da função;

- *currentReg* - armazena o número de registos atualmente em uso. Este termo é usado quando se aplica o algoritmo de alocação de registos simples;

- *tempToReg* - este termo permite estabelecer a relação entre cada temporário e o registo que lhe foi atribuído após a aplicação do algoritmo de alocação de registos;

- *regInUse* - armazena a lista de registos em uso antes de uma instrução *call*;

- *fArgs* - armazena os registos atribuídos aos temporários correspondentes aos argumentos da função numa instrução *call*.

[1] O *assembler* recolhe todas as secções *.data* e *.text* no programa e junta-as numa única secção correspondente (*.data* ou *.text*).

Além dos termos auxiliares apresentados anteriormente, foram também criados alguns predicados auxiliares. Estes predicados auxiliares são:

- *buildRec* - este predicado auxiliar permite criar o registo de ativação da função dada. Para tal, processa cada um dos elementos das listas de argumentos e variáveis locais fornecidas nos predicados *id* e armazena-as num termo global do gerador de código. A informação que consta neste termo global pode ser acedida sempre que necessário;

- *offset* - estes predicados auxiliares permitem calcular o offset de cada argumento ou variável local, sendo este valor depois armazenado no correspondente termo global do registo de ativação;

- *alocSpace* - permite calcular quanto espaço será necessário alocar para armazenar as variáveis locais e os argumentos da função. Os valores calculados por este predicado são usados no prólogo e epílogo do programa MIPS resultante da tradução das instruções da representação intermédia;

- *listInstr* - permite processar a lista de instruções que compõe o corpo de uma função, gerando o código de cada uma delas;

- *resetReg* - permite fazer o *reset* dos predicados globais auxiliares usados durante a aplicação do algoritmo de alocação de registos;

- *incrementRegC* e *decrementRegC* - permitem incrementar e decrementar o número de registos atualmente em uso, respetivamente;

- *updateR* - permite atualizar o número de registos atualmente em uso;

- *resetAll* - permite reinicializar os termos globais auxiliares que são usados na geração de código de uma função, nomeadamente: o termo correspondente à função atual, os termos que armazenam os valores relativos ao espaço alocado para os argumentos e variáveis locais dessa função e os termos compõem o registo de ativação da função;

- *resetOff* - permite fazer o *reset* aos termos auxiliares que dizem respeito aos *offsets* das variáveis locais e argumentos da função (*fpOffL e fpOffA*);

- *getOffset* - permite obter o *offset* de uma variável local ou argumento da função, consultando para isso o registo de ativação da função respetiva;

- *getRegArgs* - permite obter os registos correspondentes aos temporários da lista de argumentos de uma chamada de função ou procedimento;

- *pushArgs* - permite gerar o código correspondente ao *push* dos argumentos na geração de código de uma chamada de função ou procedimento;

- *saveRegInUse* - permite gerar o código correspondente ao armazenamento dos valores dos registos em uso quando é feita uma chamada de função ou procedimento;

- *used* - permite determinar todos os registos atualmente em uso;
- *removeArgs* - permite remover da lista de todos os registos atulamente em uso os argumentos usados na chamada de uma função ou procedimento;
- *getReg* - permite obter a lista final de registos atualmente em uso (já sem os que são usados na chamada de funções ou procedimentos);
- *restoreRegInUse* - permite gerar o código correspondente ao restauro dos valores dos registos em uso após a geração do código da chamada de uma função ou procedimento;
- *clearInfoCall* - permite fazer o *reset* aos termos globais auxiliares usados durante a geração de código de uma chamada de função ou procedimento;
- *generateMacros, generatePIMacro e generatePBMacro* - permitem gerar o código correspondente às *macros* das instruções *print*.

4 Aspetos Práticos

A geração do código de uma representação intermédia pode ser feita de duas formas. A primeira forma consiste em usar o comando:

```
swipl -f A < B > C
```

Em que A corresponde ao ficheiro *source* (no caso deste trabalho, esse ficheiro será o *codegen.pl*), B corresponde ao ficheiro da representação intermédia para a qual se pretende fazer a geração de código (por exemplo, *twice.pl*) e C corresponde ao ficheiro que irá armazenar o código gerado pelo programa (por exemplo, *twice.asm*). Esta forma é rápida e simples, porém tem um problema: se existirem termos que não sejam separados por uma linha vazia, ocorre um erro.

Em alternativa a este modo de gerar a representação intermédia, foram criados dois predicados. O predicado *main*, permite ler do *standard de input* as instruções da representação intermédia e gerar o código para as mesmas, que é depois apresentado no *standard de output*. O predicado *exit*, permite sair do *main*.

5 Conclusão

O gerador de código implementado foi testado com a representação intermédia das funções fornecidas juntamente com o enunciado do trabalho. Os resultados obtidos foram consistentes e, usando como comparação os ficheiros .asm fornecidos em conjunto com o enunciado do trabalho, parecem corretos. Trabalho futuro poderia incluir reduzir os predicados e termos auxiliares em excesso e melhorar a implementação.

Referências

[1] V. Pedro, "Instruction Selection." https://www.moodle.uevora.pt/1415/pluginfile.php/31234/mod_resource/content/4/is-v3.pdf, 2014.