UNIVERSIDADE ESTADUAL DE MARINGÁ
Departamento de Informática

Laboratório de Compiladores

Compiladores
Prof.: Anderson Faustino da Silva

(Versão 0.1)

1. Introdução

Este trabalho prático tem por objetivo desenvolver um compilador para a Linguagem GCL [1], capaz de gerar código executável para uma arquitetura de 32 bits.

O compilador deve conter as seguintes fases:

01) Análise Léxica
02) Análise Sintática
03) Análise Semântica
04) Tradução para Código Intermediário
05) Otimização (*inline*)
06) Seleção de Instrução
07) Alocação de Registradores
08) Emissão de Código Executável

2. A Implementação

O compilador deverá ter as seguintes opções:

-da Emitir árvore sintática abstrata
-di Emitir código intermediário
-ds Emitir código após seleção de instruções
-s Emitir código *assembly*
-i Ativa a otimização *inline*
-o Nome do arquivo de saída
-h Help

Em síntese:

gclc [-d[ais]] [-s] [-i] [-h] [-o <nome>] <*prog.gcl*>

A saída será um arquivo com código executável. Não é necessário implementar um montador, basta gerar código *Assembly* e fazer uma chamada ao GAS [2] (de forma automática, ou seja, transparente ao usuário). Caso não seja utilizada a opção **-o**, o nome do arquivo de saída será **a.out**.

- **Extensões da Linguagem**
  DEVE ser adicionado a linguagem:
  1. Tipo *string*
  2. Tipo real
  3. Números com sinal

- **Análise Léxica**
  Esta fase deve ser implementada na forma de um autômato. A implementação NÃO DEVE UTILIZAR ferramentas geradoras de analisadores léxicos.

- **Análise Sintática**
  O analisador sintático deverá ser LALR e utilizar um autômato na implementação. A implementação NÃO DEVE UTILIZAR ferramentas geradoras de analisadores sintáticos.

- **Análise Semântica**
  O analisador deverá emitir mensagens claras quando ocorrerem erros. Deve ser emitido a linha, a coluna e uma mensagem indicando qual erro ocorreu. Além disto, o analisador deverá ser capaz de analisar declarações recursivas e o uso de *breaks*.

- **Tradução para Código Intermediário**
  Pode ser utilizada a representação intermediária proposta por Appel [3,4].

- **Otimização**
  Implementação da otimização *inline*, para qualquer tipo (e quantidade) de parâmetro.

- **Seleção de Instrução**
  Implementação da fase de geração de código x86, para o compilador gclc. A implementação DEVE utilizar um *BOTTON-UP REWRITE SYSTEM*, como por exemplo o Iburg [5].

- **Alocação de Registradores**
  Implementação de um módulo *liveness* e um *módulo para alocação de registradores*. O alocador de registradores DEVE utilizar um algoritmo de coloração de grafo. Além disto, ele DEVE implementar *spilling* e *coalescing*. Pode ser utilizado o algoritmo proposto por Appel [3,4].

- **Emissão de Código Executável**
  Implementação de um módulo para emissão de código final (executável). A implementação DEVE fazer automaticamente uma chamada ao ***as*** e ao ***ld***.

- **Deve ser implementado quatro conjuntos de programas para a avaliação do trabalho.** Os conjuntos (Cx) são:
  C1. Programas contendo os possíveis (todos) erros léxicos
  C2. Programas contendo os possíveis (todos) erros sintáticos
  C3. Programas contendo os possíveis (todos) erros semânticos
  C4. Programas sem erros, a saber:
    - Busca binária (N elementos)
    - Heap sort (N elementos)
    - N-rainhas
    - Merge sort (N elementos)
    - Busca em árvore binária (N elementos, recursivo)
    - Balanceamento de árvore binária (N elementos, recursivo)
    - N-body
    - *Red-Black Successive Over-Relaxation* (SOR) (NxM elementos)
    - Decomposição LU (NxM elementos)
    - *International Data Encryption Algorithm* (IDEA) (N elementos)

3. Avaliação do Trabalho

- O trabalho deve ser apresentado ao professor, trabalho não apresentado será anulado.

- O trabalho será avaliado de acordo com os seguintes critérios:
  1. Corretude: será avaliada a corretude do compilador implementado (por meio de uma bateria de experimentos).
  2. Completude: será avaliado se a implementação está completa (por meio de uma bateria de experimentos).
  3. Codificação: será avaliado a modularidade e a elegância do código fonte.

- Cálculo da primeira nota (análises léxica, sintática e semântica):
  Nota** = **(T1 + T2 + (T3*3) + (T4*2) + M)/8
  T1: Programas contendo erros léxicos (C1)
    - Avaliação da análise léxica

  T2: Programas contendo erros sintáticos (C2)
    - Avaliação da análise sintática

  T3: Programas contendo erros semânticos (C3)
    - Avaliação da análise semântica

  T4: Programas sem erros (C4)
    - Avaliação da opção -da

  M: modularidade

- Cálculo da segunda nota (compilador completo)
  Nota = (T5 + T6 + (T7*2) + (T8*3) + (T9*6) + M)/14
  T5: Programas sem erros (C4)
    - Avaliação da geração de código intermediário (opção -di).

  T6 = T5
    - Avaliação da fase de seleção de instruções (opção -ds)

T7 = T5

- Avaliação da fase de otimização (opção -i)

T8 = T5

- Avaliação da fase de emissão de código (opção -s).

T9 = T5

- Avaliação da geração de código executável (opção -o). Será avaliado se os programas são executáveis em uma arquitetura IA32.

***M*: modularidade**

- Cada nota parcial (Tx) terá valor entre 0 e 10 e será calculada por meio de regra de três simples.

- A quantidade de programas dos conjuntos C1, C2 e C3 deve ser especifica por cada aluno (juntamente com a implementação de cada programa do conjunto). Somente será considerada nota completa para uma determinada fase SE a fase estiver funcionando corretamente E o conjunto estiver especificado corretamente (quantidade e programas).

3. Considerações Gerais

- **O trabalho pode ser feito em dupla.**
- **Embora o trabalho seja em dupla a nota é individual, ou seja, um integrante poderá tirar nota máxima enquanto outro ficar sem nota.**
- **Data de entrega da primeira parte:** 16/09/2011 até as 23:00.
- **Data da apresentação da primeira parte:** aula após a data de entrega.
- **Data de entrega do trabalho completo:** 25/11/2011 até as 23:00.
- **Data da apresentação do trabalho completo:** aula após a data de entrega.
- **O que entregar:** enviar por *email* um pacote contendo:
  1. Identificação da dupla
  2. O código fonte da implementação
  3. Os programas teste
  4. Um manual de instalação e uso do compilador
- **Cópias de qualquer tipo ANULARÃO o trabalho.**

4. Referências

[1] Guarded Command Language, http://csis.pace.edu/~bergin/compiler/gcl.html

[2] GNU Assembler. http://sourceware.org/binutils/docs-2.20/as/index.html.

[3] APPEL, A. W. Modern Compiler Implementation in C/Java/ML. Cambridge, 1998.

[4] Modern Compiler Implementation http://www.cs.princeton.edu/~appel/modern/.

[5] IBURG: A Tree Parser Generator, http://code.google.com/p/iburg/.