**1º Trabalho Prático**
CIC 116432 – Software Básico
Prof. Bruno Macchiavello
1º Semestre de 2014

# 1 Introdução

O trabalho consiste em duas partes: (*i*) implementar em C/C++ um método de tradução de uma linguagem de montagem simples para uma representação de código objeto, (*ii*) implementar um programa em C/C++ para simular a execução de um código objeto da parte anterior.

# 2 Objetivo

Fixar o funcionamento de um processo de tradução. Especificamente as etapas de analise léxica, sintática e semântica e a etapa de geração de código objeto.

# 3 Especificação

## 3.1 Montador

A linguagem de montagem utilizada será a linguagem simbólica hipotética apresentada em sala. Esta linguahem é formadapor um conjunto de apenas 14 instruções. Uma diferença com o formato visto em sala de aula é que os programas devem ser divididos em seções de código e dados.

Para cada instrução da máquina hipotética, a Tabela 1 abaixo contém o mnemônico, quantidade de operandos, código de operação utilizado na montagem, tamanho em palavras da instrução montada e uma breve descrição da sua utilidade. As linhas finais da tabela definem as diretivas para alocação de memória no segmento de dados.

Os identificadores de variáveis e rótulos são limitados em 100 caracteres e seguem as regras comuns da linguagem C, sendo compostos por letras, números ou o caractere _ (*underscore*) e com a restrição de que o primeiro caractere não pode ser um número.

Para eliminar ambiguidade, as seções de código e dados devem ser devidamente marcadas com as diretivas correspondentes e aparecer no máximo uma vez mas **em qualquer ordem**, como ilustra o exemplo abaixo:

```
SECTION TEXT
ROT: INPUT N1
     COPY N1, N4 ; comentario qualquer
     COPY N2, N3[0]
     COPY N3[0], N3[1]
     OUTPUT N3[1]
     STOP

SECTION DATA
N1:  SPACE
N2:  CONST -0x10
N3:  SPACE 2
N4:  SPACE
```

A linguagem hipotética não é sensível ao caso, não havendo diferenciação entre maiúsculas e minúsculas.

O programa de tradução deve ser capaz de realizar as fases de análise e síntese, mantendo informação intermediária armazenada em estruturas de dados. A escolha apropriada de estruturas de dados faz parte do escopo do trabalho. Quando um error ocorrer o programa deve ser interrompido e indicar ao programador a número da linha do programa com erro e o tipo de erro (léxico, sintático ou semântico). O tratamento de erros deve abranger a lista abaixo:

- Erros léxicos causados por:
  - Caracteres inválidos nos tokens de entrada;
  - Instrução ou diretiva inválida;
- Erros sintáticos causados pelo mal uso de alguma instrução ou diretiva;
- Erros semânticos causados por:
  - Declarações ausentes ou repetidas;
  - Desvios condicionais e incondicionais para endereços na seção de dados;
  - Símbolos na seção de código como operandos de dados;
  - Escrita em memória reservada para armazenamento de constantes;
  - Divisão por constantes inicializadas com valor zero

O programa de tradução deve receber três argumentos em linha de comando, um arquivo de entrada contendo um programa em *Assembly* em formato texto (.asm) na linguagem hipotética, um arquivo de saída do pré-processamento (.pre) e um arquivo de saída para armazenar o código objeto resultante (.obj). Inicialmente o programa deve fazer o pré-processamento evaluando somente as diretivas EQU e IF, e removendo os comentários. Exemplo:

**Arquivo de Entrada:**

```
SECTION TEXT
IF L1
LOAD SPACE ;faz esta operação se L1 for verdadeiro
IF L2
INPUT SPACE ;faz esta operação se L2 for verdadeiro

SECTION DATA
L1:  EQU 1
L2:  EQU 0
N: SPACE
```

**Arquivo de Pré-processado:**

```
SECTION TEXT
LOAD SPACE

SECTION DATA
N: SPACE
```

Gerando o arquivo de saída de pre-processamento em formato texto. Depois deve utilizar esse arquivo como entrada para gerar o código objeto em formato binário. O formato do código objeto gerado pelo processo de tradução deve ser a simples concatenação de todos os inteiros de 16 *bits* que representam as instruções e seus operandos na seção de código, seguido pela seção de dados contendo todas as variáveis (vetores ou não) e constantes de 16 *bits* declaradas no programa de entrada. Por convenção e apesar da diretiva `SPACE` não garantir a inicialização de variáveis, porém utilizaremos o valor 0 para iniciar o valor de todas as posições de memória alocadas com essa diretiva. A dupla pode escolher fazer o algoritmo de duas passagens ou passagem única.

## 3.2 Simulador

O programa de simulação deve receber um único argumento em linha de comando, o arquivo objeto gerado pelo programa de tradução. A simulação deve ser capaz de conservar o funcionamento correto do programa de entrada e encerrar execução na primeira ocorrência de uma instrução `STOP`. Os únicos erros tratados pelo programa de simulação são a divisão por zero e o fornecimento de valores inválidos para a instrução `INPUT`.

# 4 Avaliação

O prazo de entrega do trabalho é 12 de Outubro de 2014. A entrega consistirá em:

- Código-fonte completo e comentado com instruções de compilação dos programas de tradução e simulação;
- Programas de exemplo que demonstrem o funcionamento correto dos programas de tradução e simulação.

A forma de entrega é pelo Moodle. O trabalho pode ser feito individualmente ou em dupla.

# 5 Dicas

Algumas funções recomendadas da biblioteca padrão da linguagem de programação C que podem reduzir substancialmente o trabalho envolvido são `fread()` e `fwrite()` para leitura e escrita do código intermediário e `strtok()` para implementação do analisador léxico. Para o simulador usar um array do tipo *short* para simular a memória principal do computador.

Tabela 1: Intruções e diretivas.

| Instruções | | | | |
|---|---|---|---|---|
| Mnemônico | Operandos | Código | Tamanho | Descrição |
| `ADD` | 1 | 1 | 2 | `ACC <- ACC + MEM[OP]` |
| `SUB` | 1 | 2 | 2 | `ACC <- ACC - MEM[OP]` |
| `MULT` | 1 | 3 | 2 | `ACC <- ACC * MEM[OP]` |
| `DIV` | 1 | 4 | 2 | `ACC <- ACC / MEM[OP]` |
| `JMP` | 1 | 5 | 2 | `PC <- OP` |
| `JMPN` | 1 | 6 | 2 | Se `ACC < 0, PC <- OP` |
| `JMPP` | 1 | 7 | 2 | Se `ACC > 0, PC <- OP` |
| `JMPZ` | 1 | 8 | 2 | Se `ACC = 0, PC <- OP` |
| `COPY` | 2 | 9 | 3 | `MEM[OP2] <- MEM[OP1]` |
| `LOAD` | 1 | 10 | 2 | `ACC <- MEM[OP]` |
| `STORE` | 1 | 11 | 2 | `MEM[OP] <- ACC` |
| `INPUT` | 1 | 12 | 2 | `MEM[OP] <- STDIN` |
| `OUTPUT` | 1 | 13 | 2 | `STDOUT <- MEM[OP]` |
| `STOP` | 0 | 14 | 1 | Encerrar execução. |
| Diretivas | | | | |
| `SECTION` | 1 | - | 0 | Marcar início de seção de código (`TEXT`) ou dados (`DATA`). |
| `SPACE` | 0/1 | - | variável | Reservar 1 ou mais endereços de memória não-inicializada para armazenamento de uma palavra. |
| `SPACE` | 1 | - | N | Reservar memória não-inicializada para armazenamento de um vetor do tamanho especificado. |
| `CONST` | 1 | - | 1 | Reservar memória para armazenamento de uma constante inteira de 16 *bits* em base decimal ou hexadecimal. |
| `EQU` | 1 | - | 0 | Cria um sinônimo textual para um símbolo |
| `IF` | 1 | - | 0 | Instrue o montador a incluir a **linha seguinte** do código somente se o valor do operando for 1 |