

**1º Trabalho Prático**  
CIC 116432 – Software Básico  
Prof. Bruno Macchiavello  
2º Semestre de 2014

## 1 Introdução

O trabalho consiste em implementar em C/C++ um método de tradução de uma linguagem de montagem simples para uma representação de código objeto. O tradutor a ser implementado será um Macro-Assembler da linguagem hipotética vista em sala de aula.

## 2 Objetivo

Fixar o funcionamento de um processo de tradução. Especificamente as etapas de análise 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 linguagem é formada por 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, como ilustra o exemplo abaixo:

#### SECTION TEXT

```
ROT: INPUT N1
      COPY N1, N4 ; comentario qualquer
      COPY N2, N3
      COPY N3, N3 + 1
      OUTPUT N3 + 1
      STOP
```

#### SECTION DATA

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

O montador deve ser capaz de:

- NÃO ser sensível ao caso, podendo aceitar instruções/diretivas/rótulos em maiúsculas e minúsculas.
- NÃO ter ordem fixa para as seções de TEXT e DATA, ou seja, as seções podem estar em qualquer ordem.
- Gerar um arquivo de saída em formato TEXTO (mais detalhes serão descritos a seguir).
- Desconsiderar tabulações, quebras de linhas e espaços desnecessários em qualquer lugar do código.
- A diretiva CONST deve aceitar números positivos e negativos (inteiros e hexadecimal);
- Deve ser possível trabalhar com vetores (SPACE com operando, e usar operações do tipo: LABEL + Número)
- Capacidade de aceitar comentários indicados pelo símbolo “;”
- O comando COPY deve utilizar uma vírgula e um espaço entre os operandos (COPY A, B)
- Ser capaz de aceitar MACROS (mais detalhes serão descritos a seguir).
- Poder criar um rótulo, dar quebra de linha e continuar a linha depois (o rótulo seria equivalente a linha seguinte)
- Identificar erros durante a montagem. Montado sempre o programa inteiro e mostrando na tela a(s) LINHA(S) e TIPO DOS ERROS (segundo a relação a seguir e indicar se é LÉXICO, SINTÁTICO OU SEMANTICO). O programa deve pelo menos detetar os seguintes tipos de erro:

- declarações e rótulos ausentes;
- declarações e rótulos repetidos;
- pulo para rótulos inválidos;
- pulo para seção errada;
- diretivas inválidas;
- instruções inválidas;
- diretivas ou instruções na seção errada;
- divisão por zero (para constante);
- instruções com a quantidade de operando inválida;
- tokens inválidos;
- dois rótulos na mesma linha;
- seção TEXT faltante;
- seção inválida;
- tipo de argumento inválido;
- modificação de um valor constante;

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. Não é obrigatório o uso de Hashing, nem das das tableas MNT e MDT. Assumir que as macros vão sempre estar declaradas antes de serem chamadas e dentro da seção de texto. Assumir QUE NÃO TEM MACRO QUE CHAMA OUTRA MACRO. As macros NÃO precisam receber parâmetros.

O programa de tradução deve receber 3 (três) argumentos em linha de comando (nessa ordem e OS ARQUIVOS COM EXTENSÃO): um tipo de operação, um arquivo de entrada contendo um programa em *Assembly* em formato texto (extensão “.asm”) na linguagem hipotética e um arquivo de saída (extensão “.o”). Os tipos de operação são: (i) préprocessamento, indicado pelo argumento “-p”, coloca a extensão “.pre” no arquivo e somente avalia as diretivas EQU e IF. (ii) processamento de macros indicado pelo argumento “-m”, coloca a extensão “.mcr” no arquivo e somente avalia as diretivas EQU e IF e substitue as MACROS. (iii) montagem, indicado pelo argumento “-o”, coloca a extensão “.o” realiza a montagem de programa usando o PROCESSO DE PASSAGEM ÚNICA. Como pode ser visto a saída de um tipo de operação pode ser visto como a entrada da próxima, logo o programa pode por exemplo no tipo de operação de montagem, gerar os três arquivos de saída.

Exemplo de chamada do programa em modo de avaliação de macros:

```
./montador -m myprogram.asm myprogram.o
```

Assumir que o EQU sempre vai vir no início do programa e fora das seções de Texto e Dados. Lembrar que pode ter EQU sem IF, mas assumir que IF sempre precisa de uma declaração de EQU anterior. Exemplo, do uso de IF e EQU:

#### **Arquivo de Entrada:**

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

#### **Arquivo de Pré-processado:**

```
SECTION TEXT
LOAD SPACE

SECTION DATA
N: SPACE
```

Todos os arquivos de saída devem estar em formato TEXTO. No caso do arquivo objeto, o arquivo de saída deve ser somente os OPCODES e operandos sem quebra de linha, nem endereço indicado, mas separados por espaço.

No Moodle tem arquivos exemplos a serem utilizados. Na correção, serão utilizados outros programas além dos disponibilizados.

## **4 Avaliação**

O prazo de entrega do trabalho é 23 de Outubro de 2017. A entrega consistirá em:

- Código-fonte completo e comentado com instruções de compilação e nome dos integrantes do grupo;

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

Tabela 1: Instruções e diretivas.

Instruções				
Mnemônico	Operandos	Código	Tamanho	Descrição
ADD	1	1	2	ACC $\leftarrow$ ACC + MEM[OP]
SUB	1	2	2	ACC $\leftarrow$ ACC - MEM[OP]
MULT	1	3	2	ACC $\leftarrow$ ACC * MEM[OP]
DIV	1	4	2	ACC $\leftarrow$ ACC / MEM[OP]
JMP	1	5	2	PC $\leftarrow$ OP
JMPN	1	6	2	Se ACC < 0, PC $\leftarrow$ OP
JMPP	1	7	2	Se ACC > 0, PC $\leftarrow$ OP
JMPZ	1	8	2	Se ACC = 0, PC $\leftarrow$ OP
COPY	2	9	3	MEM[OP2] $\leftarrow$ MEM[OP1]
LOAD	1	10	2	ACC $\leftarrow$ MEM[OP]
STORE	1	11	2	MEM[OP] $\leftarrow$ ACC
INPUT	1	12	2	MEM[OP] $\leftarrow$ STDIN
OUTPUT	1	13	2	STDOUT $\leftarrow$ 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.
CONST	1	-	1	Reservar memória para armazenamento de uma constante inteira de 16 <i>bits</i> em base decimal ou hexadecimal.
EQU	1	-	0	Cria um sinônimo textual para um símbolo
IF	1	-	0	Instrue o montador a incluir a <b>linha seguinte</b> do código somente se o valor do operando for 1
MACRO	0	-	0	Marcar início de suma MACRO. Sempre dentro da seção TEXT e antes do código principal
END	0	-	0	Marcar o fim de uma MACRO.