1º **Trabalho Prático**
CIC 116432 – Software Básico
Prof. Bruno Macchiavello
1º Semestre de 2015

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 Assembler da linguagem hipotética vista em sala de aula.

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 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 duas seções: uma de código (TEXT) e outra de dados (DATA).

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.

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 -5
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.
- A seção de dados deve vir depois da seção de códigos.
- Gerar um arquivo de de saída em formato TEXTO (mais detalhes serão descritos a seguir).
- Desconsiderar tabulações e espaços desnecessários em qualquer lugar do código.
- A diretiva CONST deve aceitar números positivos e negativos (inteiros, somente decimal);
- 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+espaço entre os operandos (COPY A, B)
- Identificar erros durante a montagem. Montado sempre o programa inteiro e mostrando na tela aa LINHAS e TIPO DOS ERROS (léxico, sintático, semântico). O programa deve pelo menos detetar os seguintes tipos de erro:
  - declarações ausentes;
  - declarações repetidas;
  - pulo para rótulos inválidos;
  - diretivas inválidas;

- instruções inválidas;
- diretivas ou instruções na seção errada;
- divisão por zero;
- instruções com a quantidade de operando inválida;
- tokens inválidos;
- dois rótulos na mesma linha;
- rótulos repetidos;
- seção (TEXT ou DATA) faltante;
- seção inválida;
- tipo de argumento inválido;
- endereço de memória não reservado (incluíndo tratamento de vetores, ou seja somente deve ser possível ter acesso a vetores dentro do tamano reservado para cada um deles);
- modificação de um valor constante.
- se não for um módulo deve possuir pelo menos uma instrução de STOP (podendo ter mais que uma)

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 ordenação de tabelas. O grupo pode escolher utilizar o algoritmo de duas passagens ou passagem única. O montador deve ser capaz de avaliar as diretivas EQU e IF

O programa de montagem (chamado 'montador.c') deve receber dois argumentos em linha de comando (nessa ordem): um arquivo de entrada contendo um programa em *Assembly* em formato texto (só nome, assume-se a extensão ".asm") na linguagem hipotética e um arquivo de saída (so ó nome sem extensão assume-se extensão ".o").

O arquivo objeto deve estar em formato TEXTO. Se o programa for um módulo, o arquivo de saída deve indicar a tabela de USO, tabela de DEFINIÇÕES e os OPCODES e operandos sem quebra de linha, nem endereço indicado, separadados por espaço. As diferentes informações devem estar separados por os indicadores TABLE e CODE. Além de ter uma seção indicando quais códigos são realtivos (no exemplo abaixo os códigos 1,3 e 4 que correspondem a 12,15,04 são relativos). Se não for um módulo o arquivo de saída deve ser OPCODES e operandos sem quebra de linha: nem endereço indicado, separadados por espaço, SEM O INDICADOR DE SEÇÃO "CODE", sem a indicação de absoluto e relativo, sem tabelas (ou seja, somente uma linha de números). Ao usar SPACE colocar ZERO no código máquina nos endereços reservados.

```
TABLE USE
```

```
ROT1 11
ROT1 15
ROT2 18

TABLE DEFINITION
ROT3 4

RELATIVE
1 3 4

CODE
14 12 12 15 04 14 12 5
```

3.2 Ligador

Fazer um código (ligador.c) que receba por linha de comando o nome de três arquivos (sem extensão, assume-se extensão ".o" para os dois primeiros e extensão ".e" para o terceiro). O programa deve fazer a ligação entre os dois primeiros módulos gerando o arquivo ligado de saída. O arquivo de saida deve ser em formato TEXTO contendo OPCODES e operandos sem quebra de linha, nem endereço indicado, separadados por espaço, SEM O INDICADOR DE SEÇÃO "CODE", sem a indicação de absoluto e relativo, sem tabelas (ou seja, somente uma linha de números).

No Moodle tem arquivos exemplos a serem utilizados. Na correção, serão utilizados vários outros programas além dos disponibilizados. No Moodle existe também um simulador para rodar os arquivos objetos.

4 Avaliação

O prazo de entrega do trabalho é 11 de Maio de 2016. A entrega consistirá em:

- Código-fonte completo e comentado com instruções de compilação dos programas de tradução e simulação;
- Incluir um comentário indicando os dois membros do grupo, assim como o sistema operacional onde o código foi realizado e instruções de como compilar o código.

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

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	1	-	1	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 *bits* em base decimal ou hexadecimal.
PUBLIC	0	-	0	Indica que o rótulo é público
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
EXTERN	0	-	0	Indica que o rótulo é um símbolo externo
BEGIN	0	-	0	Marcar início de um módulo
END	0	-	0	Marcar o fim de um ,ódulo.