1º Trabalho Prático
CIC 116432 – Software Básico
Prof. Bruno Macchiavello
1º Semestre de 2017

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 analise léxica, sintática e semântica, etapa de geração de código objeto e ligação.

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 50 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.
- 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 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 SEM 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 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, a não ser que seja utililzado as diretivas BEGIN e END. Nesse caso não deve ser verificado este erro)
- Falta de diretive END ou BEGIN (caso exista a outra)
- Erros de ligação (verificar parte de ligador)

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. Pode ser utilizado o algoritmo de passagem única ou de duas passagems. O programa não pode depender de arquivos anexos ou pastas específicas.

O programa de tradução deve receber três argumentos em linha de comando (nessa ordem): um tipo de operação, um arquivo de entrada contendo um programa em *Assembly* em formato texto (verificar que estensão seja ".asm" ou ".pre") na linguagem hipotética e um arquivo de saída (SO O NOME DO ARQUIVO DE SAÍDA SEM ESTENSÃO). Os tipos de operação são: (i) preprocessamento, indicado pelo argumento "-p", recebe como entrada um arquivo ".asm" e gera um arquivo de saída com estensão ".pre" e somente avalia as diretivas EQU e IF. (ii) motagem, indicado pelo argumento "-o", receve um arquivo de entrada ".asm" ou ".pre" e coloca a estensão ".o" na saída e eealiza a montagem de programa. 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 dois arquivos de saída. Ou seja, se chamar a opção "-o" e a entrada for ".asm", o programa pode internamente chamar o pre-processador primeiro. No caso da diretiva EQU ela deve ser sempre utilizada no inicio do código. O montador deve indicar o erro e o tipo dele: se é lexico, sintático ou semantico.

Exemplo, do uso de IF e EQU:

Arquivo de Entrada:

```
L1:  EQU 1
L2:  EQU 0

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

SECTION DATA
N: SPACE
```

Arquivo de Pré-processado:

```
SECTION TEXT
LOAD N

SECTION DATA
N: SPACE
```

Todos os arquivos de saída devem estar em formato TEXTO. No casso do arquivo objeto, o arquivo de saída deve ser somente os OPCODES e operandos sem quebra de linha, nem endereço indicado, mas sepradados por espaço (SEM INDICADOR "CODE").

Se foram usadas as diretivas BEGIN e END o arquivo de saída deve indicar a tabela de USO, tabela de DEFINIÇÕES, Informação de realocação e os OPCODES e operandos sem quebra de linha, nem endereço indicado, sepradados por espaço. A informação de realocação pode ser por mapa de bits ou lista de endereços. As diferentes informações devem estar separados por os indicadores TABLE e CODE. Exemplo de um arquivo a ser ligado (não relacionado ao exemplo anterior) pode ser visto abaixo:

```
TABLE USE
ROT1 11
ROT1 15
ROT2 18

TABLE DEFINITION
ROT3 4

TABLE REALOCATION
0101000

CODE
14 12 12 15 04 12 5
```

3.2 Ligador

Fazer um código (ligador.c) que receba por linha de comando o nome de até 4 arquivos (a extensão ".o" deve estar presente em todos menos o último arquivo, o último deve ser extensão ".e"). O programa deve fazer a ligação entre os dois ou três 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, sepradados por espaço, SEM O INDICADOR DE SEÇÃO "CODE", sem a indicação de absoluto e relativo, sem tabelas (ou seja, somente uma linha de números).

O ligador deve verificar durante o processo de ligação, se ficaram símbolos não definidos.

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 22 de Maio de 2017. A entrega consistirá em:

- Código-fonte completo e comentado dos programas de tradução e ligação. Deve ser entregue também um arquivo TEXTO indicando detalhadamente a forma de compilar os programas. A execução dos programas DEVE serguir o formato indicado no trabalho. Neste arquivo texto deve ser indicado também os nomes e matrículas dos integrantes do grupo.

A forma de entrega é pelo Moodle. O trabalho pode ser feito individualmente ou em dupla. Somente um dos integrantes deve entragar o trabalho.

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.
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
PUBLIC	0	-	0	Indica que o rótulo é público
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.