

1º Trabalho Prático
CIC 116432 – Software Básico
Prof. Bruno Macchiavello
1º Semestre de 2020

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 (algoritmo de 2 passagens e passagen única), 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 e diretivas de pre-processamento.

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 (seção de dados DEVE vir DEPOIS da seção de texto, pode assumir que isso sempre será verdade):

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

SECTION DATA
N1: SPACE
N2: CONST -48
N4: SPACE
```

O montador deve ser capaz de:

- NÃO ser sensível ao caso, podendo aceitar instruções/diretivas/rótulos em maiúsculas e minúsculas.
- Desconsiderar tabulações, quebras de linhas e espaços desnecessários em qualquer lugar do código.
- A diretiva CONST deve aceitar números positivos e negativos (somente em decimal).
- Capacidade de aceitar comentários indicados pelo símbolo ";" em qualquer lugar do código
- O comando COPY deve utilizar uma vírgula e um espaço entre os operandos (COPY A, B)
- Poder criar um rótulo, dar quebra de linha e continuar a linha depois (o rótulo seria equivalente a linha seguinte)

O programa de tradução deve ser capaz de realizar as fases de análise e síntese. O programa deve chamar "montador" e receber o arquivo de entrada por argumento na linha de comando (ex: ./montador -p myprogram.asm). Se é utilizado o argumento "-p" o programade deve somente pre-processar e dar como saída o arquivo com o mesmo nome MUDANDO a extensão para .PRE. Se é utilizado o argumento "-o" o programade deve somente pre-processar e dar como saída o arquivo com o mesmo nome MUDANDO a extensão para .OBJ. E obrigatório que o nome do programa

seja o indicado e o funcioamaneto seja também conforme indicado, caso contrário o trabalho recebe automaticamente a nota ZERO. Ambos arquivos de saída devem estar em FORMATO TEXTO (não em arquivos binarios).

Assumir que o EQU sempre vai vir no inicio 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
```

3.2 A escolher:

Escolher uma das opções abaixo:

- Você pode escolher se o montador vai aceitar a diretiva MACRO ou não. Caso positivo, o montador deve avaliar e expandir as macros durante pre-processamento. Assumir que no máximo serão declaradas 2 macros por programa, que nunca uma macro vai chamar outra macro (ou possuir qq diretiva dentra da macro) e que as macros podem receber de 0 a 2 argumentos. Ou,
- 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 tipo de linha deve ser me relação ao arquivo pre-processado para facilitar (mas se desejar pode ser com relação ao original). O programa deve pelo menos detetar os seguintes tipos de erro:
 - declarações e rótulos ausentes;

- declarações e rótulos repetidos;
- diretivas inválidas;
- instruções inválidas;
- diretivas ou instruções na seção errada;
- instruções com a quantidade de operando errado;
- instruções com o tipo de operando inválido;
- tokens inválidos;
- dois rótulos na mesma linha;
- seção TEXT faltante;
- seção inválida;

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 caso da diretiva SPACE deve ser colocado o valor OO no arquivo objeto (não colocar XX).

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

Nota: não serão modificados os arquivos de teste para adaptar ao programa desenvolvido pelo grupo. Por exemplo, foi especificado que o COPY deve receber operando separados por vírgula e SEM espaço entre os operandos. Se a grupo indica que o programa deles funciona com ESPAÇO entre os operandos, o arquivo de teste NÃO sera modificado e o teste será dado como FALHO.

4 Para entregar

Para entregar o trabalho ele deve ser entregue num uma arquivo pelo MOODLE. Junto com um README.TXT. No readme deve estar o nome e matrícula do aluno. Assim, como instruções de compilação do código. Deve tamber indicar se foi implementado a MACRO ou a detecção de erros.

Tabela 1: Instruções 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	Reservar 1 endereço 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	Instruir o montador a incluir a linha seguinte 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
ENDMACRO	0	-	0	Marcar o fim de uma MACRO.