Trabalho Prático 1
CIC 116432 – Software Básico
Prof. Bruno Macchiavello

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 passagem 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. Vc pode escolher entre uma os duas passagens.

Para cada instrução da máquina hipotética, deve-se usar a Tabela apresentada em sala de aula a qual contém o mnemônico, quantidade de operandos, código de operação (OPCODE) utilizado na montagem, tamanho em palavras da instrução montada. Além disso, o montador deve ser capaz de aceitar as diretivas SPACE e CONST como vista em sala de aula. SPACE pode receber UM argumento.

Os identificadores de variáveis e rótulos são limitados em 30 caracteres, simplesmente pq NUNCA vai ter um label maior que isso. Os rótulos 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 seçoes de código e dados devem ser devidamente marcadas com as diretivas correspondentes, como ilustra o exemplo abaixo.
Seção de dados pode vir em qualquer antes ou depois da seção de texto no arquivo de entrada.

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


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

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 SEM espaço entre os operandos (COPY A,B) Da mesma foma rótulo de array não tem vírgula entre e a constante (ADD X+2).
- 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 (com extensão) por argumento na linha de comando (ex: ./montador myprogram.asm). O programa deve dar como saída o arquivo com o mesmo nome MUDANDO a extensão para .OBJ. É obrigatório que o funcionamento do programa seja tal como descrito, caso contrário o trabalho recebe automaticamente a nota ZERO.

O arquivo de saída deve estar em FORMATO TEXTO (não em arquivos binário). Sendo que deve ser uma única linha com espaço entre OPCODE e operandos. No caso da diretiva SPACE deve ser colocado o valor 00 ou 0 (dois ou um digitos zero) no arquivo objeto (não colocar XX). Como detalhada na seção 3.2 a seção de dados no arquivo OBJETO deve ir sempre depois da seção de texto. Lembrando que no arquivo objeto não existe separação entre seções. Tudo é uma única linha separada por espaços.

O montador deve identificar os seguintes erros durante a montagem. Montado sempre o programa inteiro e mostrando na tela a(s) LINHA(S) do arquivo original e TIPO DOS ERROS (LEXICO, SINTATICO ou SEMANTICO):

1. declarações e rótulos ausentes;
2. declarações e rótulos repetidos;
3. diretivas e instruções inválidas;
4. instruções e diretivas com a quantidade de operando errada;

No Moodle tem arquivos exemplos a serem utilizados. Na correção serão utilizados vários programas além dos disponibilizados no Moodle. Nota: não serão modificados os arquivos de teste para adaptar ao programa desenvolvido pelo grupo. Por exemplo, foi especificado que o COPY deve receber argumentos SEM espaço entre os operandos, o arquivo NÃO será modificado para incluir o espaço para poder rodar.

3.1 Macro

Caso o arquivo de entrada tenha extensão ".mcr". Então o montador deve chamar uma passagem de pre-processamento. Onde devem processar as macros. Não é necessário existir as tabelas MNT e MDT. Somente poderão existir no máximo 2 (duas) macros no mesmo código. Uma macro pode chamar a outro, nesse caso a que será chamada sempre será definida primeiro. As macros serão definidas antes de qualquer sessão usando as diretivas vistas em sala de aula.

Os macros podem receber no máximo dois argumentos usando a mesma sintaxe dos slides da disciplina.

Nese caso o montador deve dar duas saidas uma saida .pre que é o proframa SEM as macros. E a saida de arquivo objeto

3.2 Simulador

A segunda parte é fazer um simulador. O simulador deve chamar "**simulador**" e receber o arquivo de entrada (com extensão) por argumento na linha de comando (ex: ./montador myprogram.obj). A entrada do simulador é a saída do montador. O próprio simulador do aluno será utilizado na correção do trabalho para verificar se os programas foram traduzidos corretamente. Porém, caso não seja entregue esta parte, será utilizado um simulador padrão que sempre espera que o código referente a parte de dados esteja depois da parte de texto.

O simulador deve executar o programa OBJETO. Mostrando na tela depois de cada instrução (não diretiva). O valor do acumulador, o valor do contador de programa, e qualquer informação que seja mostrada pela instrução OUTPUT na mesma linha.E esperar digitar ENTER para executar a proxima linha.

Por exemplo, vamos assumir que num determinado momento do programa o valor do ACUMULADOR é 5, o conteúdo do rótulo DOIS é o valor 2, e o valor do contador de programa

antes de começar a executar as linhas abaixo é 20. Nesse momento devem ser executadas as seguintes linhas:

```
MULT DOIS
STORE N1
OUTPUT N1
LOAD DOIS
```

O simulador deve mostrar na tela:

- Depois da instrução MULT DOIS:
PC <- 22 ACC <- 10
- Depois da instrução
STORE N1
PC <- 24 ACC <- 10
- Depois da instrução
OUTPUT N1
PC <- 26 ACC <- 10 SAIDA: 10

O simulador deve indicar SEGMENTATION FAULT se o PC em algum momento recebe um endereço que leva a seção de dados. Deve indicar segmentation fault depois de mostrar o PC, ACC e OUTPUT como indicado anteriormente. E depois terminar de simular.