Trabalho Prático 1
Software Básico
Prof. Bruno Macchiavello
1 o Semestre de 2022

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.

O trabalho pode (e recomenda-se) ser feito em grupo de 2 alunos. Deve ser entregue somente o código e um arquivo README indicando o SO utilizado e como compilar o programa. Sendo que se for LINUX deve-se utilizar o GCC, se for Windows deve ser o CODEBLOCKS. Apple OS será tratado como programa em LINUX.

2 Objetivo

Fixar o funcionamento de um processo de tradução. Especificamente montagem 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. As instruções devem utilizar os mnemônicos, opcodes e quantidade de operando igual ao visto em sala de aula. As diretivas utilizadas serão: SPACE (sem argumentos), CONST, BEGIN, END, EXTERN, PUBLIC, EQU e IF.. Os identificadores de variáveis e rótulos são limitados em 99 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.

O código deve ser dividido em duas seções (diferente do visto em sala de aula) SECAO TEXTO e SECAO DADOS. As diretivas SPACE e CONST devem ser colocadas na seção DADOS que deve ir sempre depois da seção de TEXTO. Nos arquivos de teste a seção de dados será sempre depois da seção de texto, logo isso não precisa ser verificado. No código objeto deve ser colocado o valor 0 (zero) nos espaços reservados por SPACE, não XX como visto em sala de aula (ver exemplo no MOODLE). E deve ser possível fazer comentários utilizando “;” em qualquer parte do código.

```
SECAO TEXTO
ROT: INPUT N1
```

```
COPY N1, N4 ;comentário qualquer
COPY N2, N3
COPY N3, N3
OUTPUT N3
SECAO DADOS
N2: CONST -48
N4: SPACE
N3: SPACE
N1: SPACE
```

O montador deve ser capaz de:

- NÃO ser sensível ao caso, podendo aceitar instruções/diretivas/rótulos em maiúsculas e minúsculas.
- Desconsiderar tabulação, quebras de linhas e espaços desnecessários em qualquer lugar do código.
- A diretiva CONST deve aceitar números positivos, negativos e hexadecimal (0xF1).
- 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 sıntese. O programa deve chamar “montador” e receber o arquivo de entrada por argumento na linha de comando (ex: ./montador -p myprogram.asm saida.obj). O executável deve receber 3 argumentos, os dois últimos são o nome de entrada e saída respectivamente. O primeiro parâmetro indica o funcionamento do montador.

-p: pre-processar as diretivas EQU e IF
-o: traduzir o código utilizando o algoritmo de duas ou uma passagem

Assumir que o EQU sempre vai vir no início do programa fora da seção de texto, caso a diretiva for utilizada (primeiras linhas do código). Pode ter mais de um EQU. Lembrar que pode ter EQU sem IF, mas assumir que IF sempre precisa de uma declaração de EQU anterior. Ou seja, o IF somente será utilizado com um rótulo que foi definido previamente por EQU. Mas o EQU não precisa ser utilizado específicamente para o IF.

```
L1: EQU 1
L2: EQU 0
SECAO TEXTO
IF L1
LOAD SPACE ;faz esta operação se L1 for verdadeiro
IF L2
INPUT SPACE ;faz esta operação se L2 for verdadeiro
```

A saída do pre-processamento deve ser um arquivo de texto pre-preprocessado. Que vai ser a entrada para o opção -o.

A detecção dos erros abaixo deve ser feita somente quando utilizado a opção “-o”. Não serão testados erros de pre-processamento. Deve indicar o erro com relação a linha do arquivo pre-processo (caso seja necessário pre-processar). Indicar o tipo de erro (LÉXICO, SINTÁTICO OU SEMÂNTICO).

– declarações rótulos ausentes;
– declarações ou rótulos repetidos;
– instruções com a quantidade de operando errado;
– tokens inválidos;
– dois rótulos na mesma linha;
- instruções ou diretivas nas seções erradas;
- falta de seção de texto.

Os arquivos objeto devem estar em 2 formatos. Caso o arquivo de entrada NÃO tenha BEGIN e END. o arquivo objeto deve ser dado em formato texto numa única linha:

12 29 10 29 4 28 11 30 3 28 11 31 10 29 2 31 11 31 13 31 9 30 29 10 29 7 4 14 2 0 0 0

Caso os arquivos tenham BEGIN e END. O Formato deve ser:

TABELA USO
Y 5
Y 9
L1 3

TABELA DEF
VAR 29

12 29 10 29 4 28 11 30 3 28 11 31 10 29 2 31 11 31 13 31 9 30 29 10 29 7 4 14 2 0 0 0

3.2 Ligador

Deve ser realizado um programa que realiza a ligação de até 2 programas. As entradas devem ser passadas por linha de comando (./ligador myfile.o myfle2.o). Os arquivos objeto devem ter Tabela de uso e Definição. Somente serão utilizados 2 arquivos objetos. Deve gerar um arquivo ligado de saída. Que vai ser no mesmo formato do arquivo objeto caso NÃO tenha BEGIN e END. UM arquivo de texto em uma única linha. Para alinhar os códigos sempre deve manter a ordem dos arquivos que foi passado na linha de comando. Sempre serão USADAS as saídas do próprio montador feito pelos alunos como entrada para o ligador.