Trabalho Prático 1
Software Básico
Prof. Bruno Macchiavello
1 o Semestre de 2021

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 carregar o programa em memória e programação IA-32. 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 instruc̃oes. 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 mnemˆonico, 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 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.

As diretivas SPACE e CONST podem ser colocadas em qualquer parte do código. Porém deve colocar os espaços reservados e as constantes no final do código objeto (após o OPCODE de stop). 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.

```
ROT: INPUT N1
COPY N1, N4 ;comentario qualquer
COPY N2, N3
COPY N3, N3
N2: CONST -48
OUTPUT N3
N4: SPACE
```

STOP
N1: SPACE

O montador deve ser capaz de:

- NAO 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 e negativos (somente em decimal).
- O comando COPY deve utilizar uma v´ırgula e um espa¸co 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 (não serão usadas dentro de MACROS)
-m: pre-processar as MACROS
-o: traduzir o código utilizando o algoritmo de duas ou uma passagem

Assumir que o EQU sempre vai vir no início do programa, 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

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

O código deve ser capaz de pré-processar UMA (e somente UMA) macro por programa. Que deve ser utilizada conforme visto em sala de aula, e assumir que NÃO vai receber parâmetros. Não é necessário fazer uma tabela MNT dado que somente existe uma MACRO.

O código deve identificar erros durante a montagem. Montado sempre o programa inteiro e mostrando na tela a(s) LINHA(S), ERRO e TIPO DOS ERRO (segundo a relação a seguir e indicar se qual desses erro ocorreu e se é LEXICO, SINTÁTICO OU SEMANTICO). O tipo de

linha deve ser em relação ao arquivo pre-processado para facilitar (mas se desejar pode ser com relação ao original).

A detecção dos erros abaixo deve ser feita somente quando utilizado o opção “-o”:

– declarações rótulos ausentes;
– declarações ou rótulos repetidos;
– instruções com a quantidade de operando errado;
– tokens inválidos;
– dois rótulos na mesma linha;

A detecção dos erros abaixo deve ser detectado com a opção “-p”:

- IF com rótulo não declaro por um EQU
- EQU com rótulo não utilizado

A detecção dos erros abaixo deve ser detectado com a opção “-m”:

- Falta de ENDMACRO
- MACRO não utilizada

O programa será testado com diferentes arquivos de entrada. Os arquivos de entrada não serão fornecidos, para evitar que o código funcione somente com os programas indicados.

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				
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	Instrue 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.