1º Trabalho Prático
CIC 116432 – Software Básico
Prof. Bruno Macchiavello
2º Semestre de 2019

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 analise 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.

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 pode vir ANTES ou DEPOIS da seção de texto):

```
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.
- Gerar dois arquivos de de saída em formato TEXTO (mais detalhes serão descritos a seguir).
- Desconsiderar tabulações, quebras de linhas e espaços desnecessários em qualquer lugar do código.
- A diretiva CONST deve aceitar números positivos (inteiros e hexadecimal) e negativos (inteiros). Em hexadecimal deve ser utilizado o formato '0x'
- Deve ser possível trabalhar com vetores (SPACE com operando, e usar operações do tipo: LABEL + Número, colocando espaço entre o '+')
- Capacidade de aceitar comentários indicados pelo símbolo ";" em qualquer lugar do código
- O comando COPY deve utilizar uma vírgula e um SEM espaço entre os operandos (COPY A,B)
- Utilizar o algoritmo de passagem única.
- Poder criar um rótulo, dar quebra de linha e continuar a linha depois (o rótulo seria equivalente a linha seguinte)
- 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. O programa deve pelo menos detetar os seguintes tipos de erro:

- declarações e rótulos ausentes;
- declarações e rótulos repetidos;
- pulo para rótulos inválidos;
- pulo para seção errada;
- diretivas inválidas;
- instruções inválidas;
- diretivas ou instruções na seção errada;
- divisão por zero (para constante);
- instruções com a quantidade de operando inválida;
- instruções com o tipo de operando inválido;
- tokens inválidos;
- dois rótulos na mesma linha;
- seção TEXT faltante;
- seção inválida;
- tipo de argumento inválido;
- modificação de um valor constante;
- acessar posição não reservada pelo SPACE (exemplo accesar SPACE+4, sendo que somente foi reservada uma posição)

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 myprogram.asm). O programa deve dar duas saídas como arquivos de texto. Uma sendo o arquivo pre-processado com o mesmo nome do arquivo original e extensão "pre". E o outro o arquivo objeto, com o mesmo nome do arqivo original e extensão "obj" (Deve ser retirada a extensão original do arquivo). O nome do arquivo de entrada deve incluir a extensão. E obrigatório que o nome do programa seja o indicado e o funciomaneto seja também conforme indicado, caso contrário o trabalho recebe automáticamente a nota ZERO.

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 anteior. 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
```

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. No casso 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.

3.2 Simulador

O simulador deve ser chamado "simulador" e deve receber por argumento o arquivo objeto saída do montado (ex: ./simulador myprogram.obj). O simulador deve executar o programa. Com as seguintes características:

- Sempre que a instrução INPUT seja usada deve aparecer no monitor: "Favor, inserir um valor númerico:"
- Sempre que a opreção OUTPUT seja utilizada deve aparecer no monitor: "O valro de saída é: "
- Ao executar uma INSTRUÇÃO o programa deve mostrar sempre: (i) o conteúdo do acumulador após a operação e (ii) o endereço de memória e valor no endereço de memória de qualquer conteúdo em memória que tenha sido modificado pela instrução

A forma de entrega é pelo Moodle. O trabalho pode ser feito individualmente ou em dupla.

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