

# Trabalho Prático 2: Expansor De Macros

## 1. Introdução

O trabalho tem por objetivo criar um expansor de Macros, que será utilizado pelo montador do TP2 e, posteriormente, pela Máquina Virtual criada no Trabalho prático 1. Esse expansor é implementado para fazer a tradução de um conjunto de instruções em Assembly que contém macros declaradas para a linguagem Assembly que contém as macros expandidas. O código gerado pode ser utilizado no montador da Máquina Virtual, que receberá o código em sua entrada.

Além das instruções que serão traduzidas, este trabalho implementa ainda algumas pseudo-instruções, que são utilizadas apenas pelo expansor. Essas instruções indicam para o programa onde inicia e onde termina a declaração de uma macro. São elas : BEGINMACRO e ENDMACRO.

## 2. Implementação

Ao ser executado, o expansor deve receber 2 argumentos obrigatórios: arquivo de entrada e arquivo de saída dos dados. A compilação do programa foi feita por um Makefile. Para gerar o arquivo binário executável, basta entrar no diretório que contem o arquivo Makefile pelo terminal e digitar “make”. Assim, um executável chamado "expansor" será criado na pasta */bin*. Quando é iniciado, o expansor faz uma validação da entrada e, caso os argumentos obrigatórios não tenham sido passados na execução pelo terminal, o programa será finalizado. Dessa forma, para que o programa seja executado normalmente, é necessário executar o programa pelo terminal com o seguinte comando:

```
./expansor [arg1] [arg2]
```

O expansor é composto por dois arquivos .c: “macro.c” e “macro\_func.c”. O primeiro conta a função principal. O segundo arquivo, por sua vez, contém a implementação das funções utilizadas no expansor. Essas funções estão declaradas no arquivo “macro\_func.h”, que contém ainda a declaração de uma estrutura de dados usada para criar uma tabela de macros. Nessa tabela, estarão contidos o nome da macro, o nome do operando da macro e a posição da macro no arquivo de entrada.

Primeiramente, o expansor verifica o número de linhas contidas no arquivo de entrada. Essa verificação é feita para que alguns vetores estáticos possam ser declarados com o tamanho do

número de linhas do arquivo e para que possa ser criada uma tabela de macros com o tamanho máximo igual ao número de linhas do arquivo de entrada.

Para realizar a expansão das macros contidas no arquivo de entrada, o programa cria uma tabela de macros. Essa tabela é criada por meio da função “tabela\_macros”. Após a criação da tabela, o programa chama a função “imprime”, que realiza a impressão do código no arquivo de saída passado como argumento na execução do expensor.

A função “tabela\_macros” faz a leitura de cada linha do arquivo de entrada e verifica se a linha contém alguma macro seguida da instrução BEGINMACRO. Caso isso ocorra, uma declaração de macro é iniciada nesta linha. Nesta situação, a função armazena o nome e a linha que contém a declaração da macro em uma tabela. Essa função verifica ainda se a macro recebe algum argumento na sua declaração e, se ela receber, o argumento é armazenado também na tabela de macros.

Outra função importante no programa é a “imprime”. Essa função verifica, em cada linha do arquivo de entrada, se há declaração de macro ou não. Caso haja, as linhas do arquivo relacionadas à declaração da macro são ignoradas pela função. O número de linhas a serem ignoradas é obtido por meio da função “tamanho\_macro”. Se não houver declaração, a função verifica se o operador da instrução contida na linha é uma chamada para macro.

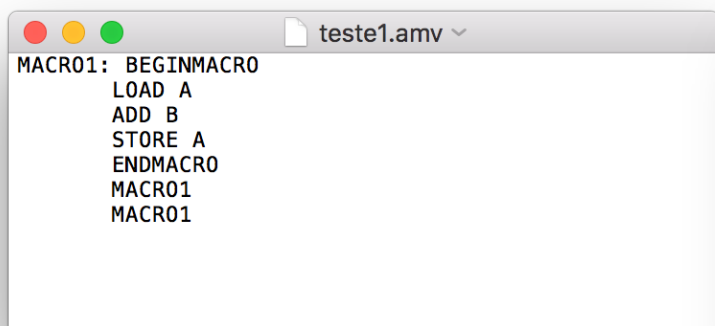
Quando não ocorre a chamada para a macro nem a declaração de uma macro, a linha é impressa no arquivo de saída. Por outro lado, se houver uma chamada de macro, o programa verifica qual é a macro e consulta sua posição na tabela. Após verificar a posição da macro, o programa imprime no arquivo de saída a expansão da macro, por meio da função “imprime\_macro”. Por fim, na expansão da macro, é verificado se o operando da instrução é um argumento da macro ou não. Caso seja, o argumento é impresso na expansão.

## 3. Testes

Seis testes foram realizados com o expensor. Os testes estão armazenados no diretório *tp2\_alefmon/tst* e foram nomeados como teste1.amv, teste2.amv, teste3.amv, fatorial.amv e mdc.amv, respectivamente. O detalhamento de cada um dos testes será realizado a seguir.

### 3.1 Teste 1

O teste 1 foi realizado para verificar o funcionamento do programa com macros declaradas sem argumento nenhum.



```
MACRO1: BEGINMACRO
        LOAD A
        ADD B
        STORE A
        ENDMACRO
MACRO1
MACRO1
```



```
LOAD A
ADD B
STORE A
LOAD A
ADD B
STORE A
```

## 3.2 Teste 2

O teste 2 foi feito com o objetivo de verificar a funcionalidade do programa quando é declarada uma macro que necessita de um argumento.

```
teste2.amv
MACRO2: BEGINMACRO Y
LOAD Y
ADD B
STORE Y
ENDMACRO
MACRO2 A
MACRO2 J
```

```
saida2.amv
LOAD A
ADD B
STORE A
LOAD J
ADD B
STORE J
```

## 3.3 Teste 3

O teste 3 utiliza o exemplo utilizado na especificação do trabalho.

```
teste3.amv
READ R0
READ R1
STORE R0 A
SUB R0 R1
LOAD R0 A
JN MB
COPY R2 R0
JMP
MB: COPY R2 R1
L: PRINT
HALT
A: WORD 0
PRINT: BEGINMACRO
WRITE R0
WRITE R1
WRITE R2
ENDMACRO
END
```

```
saida3.amv
READ R0
READ R1
STORE R0 A
SUB R0 R1
LOAD R0 A
JN MB
COPY R2 R0
JMP
MB: COPY R2 R1
L: WRITE R0
WRITE R1
WRITE R2
HALT
A: WORD 0
END
```

## 3.4 Teste 4

O teste 4 é feito com o teste disponibilizado na pasta do Trabalho. O teste tem por objetivo verificar o funcionamento do programa quando o arquivo de entrada contém um código com linhas em branco. Além disso, o código contém declarações de macro com a utilização de argumentos.

```
teste4.amv
PRINT2: BEGINMACRO
WRITE R3
WRITE R1
WRITE R2
ENDMACRO

READ R0
READ R1

STORE R0 A
SUB R0 R1
LOAD R0 A

JN MB
COPY R2 R0
JUMP L
MB: COPY R2 R1
L: PRINT R0
    HALT

A: WORD 0

PRINT: BEGINMACRO A
WRITE A
LABEL: WRITE R1
WRITE R2
ENDMACRO

PRINT R0

PRINT2
END
```

```
saida4.amv
READ R0
READ R1
STORE R0 A
SUB R0 R1
LOAD R0 A
JN MB
COPY R2 R0
JUMP L
MB: COPY R2 R1
L: WRITE R0
LABEL: WRITE R1
WRITE R2
HALT
A: WORD 0
WRITE R0
LABEL2: WRITE R1
WRITE R2
WRITE R3
WRITE R1
WRITE R2
END
```

## 3.5 Teste 5

O teste 5 é o programa “fatorial”.

```
fatorial.amv
READ R0
COPY R1 R0
COPY R2 R0
COPY R3 R0
LOAD R4 A
MULTIPLICA: BEGINMACRO
ADD R0 R1
SUB R2 R4
ENDMACRO
SUB R2 R4
SUB R3 R4
SUB R3 R4
FATORIAL: MULTIPLICA
JZ DESINCREMENTA
JMP FATORIAL
DESINCREMENTA: SUB R3 R4
JZ FIM
COPY R2 R3
JMP FATORIAL
FIM: WRITE R0
HALT
A: WORD 1
END
```

```
saida_fat.amv
READ R0
COPY R1 R0
COPY R2 R0
COPY R3 R0
LOAD R4 A
SUB R2 R4
SUB R3 R4
SUB R3 R4
FATORIAL: ADD R0 R1
SUB R2 R4
JZ DESINCREMENTA
JMP FATORIAL
DESINCREMENTA: SUB R3 R4
JZ FIM
COPY R2 R3
JMP FATORIAL
FIM: WRITE R0
HALT
A: WORD 1
END
```

## 4. Conclusão

A criação de um programa que realiza a expansão das macros é uma tarefa complicada. Essa tarefa requer um amplo conhecimento a respeito do funcionamento não só do montador, mas também de outras camadas de um computador. Toda abordagem do conteúdo feita pela disciplina de Software Básico colabora muito para a execução do trabalho.

Por fim, o Trabalho Prático 3 mostra a importância e as facilidades trazidas pelo uso de uma macro. O uso da macro permite a criação de programas com uma grande economia de linhas de código. Além da economia no tamanho do código, as macros exercem um papel fundamental na utilização de um loop. Durante a criação dos programas de teste, é evidente a diferença no nível de dificuldade da criação de um programa. Portanto, o uso das macros traz muitos benefícios ao programador, principalmente em códigos com o tamanho grande.

## 5. Referências

[1] TANENBAUM, Andrew S.; AUSTIN, Todd. Organização Estruturada de Computadores. 6. ed. : Pearson, 2013.