

Trabalho Prático 3

Software Básico

Enzo Roiz

1- Introdução

Visando entender melhor os conceitos que serão abordados durante o curso, uma série de trabalhos práticos serão desenvolvidos como uma forma de fixar tais conceitos. No primeiro trabalho foi implementado um emulador para uma máquina virtual básica. No segundo trabalho prático foi implementado um montador de dois passos. Este trabalho visa implementar um expansor de macros, permitindo que macros sejam definidas dentro do programa e seu código seja expandido onde a macro for chamada.

Assim como no montador, a expansão de macro ocorrem em dois passos. Por meio de um arquivo, passado como parâmetro para o programa, o programa verifica as definições de macro, e salva suas instruções e informações necessárias em uma primeira passagem pelo arquivo de entrada. Na segunda passagem pelo arquivo de entrada são identificadas as chamadas de macro e o código da macro correspondente é expandido, substituindo a chamada.

Além das instruções já implementadas no primeiro trabalho prático e das duas instruções criadas no segundo trabalho prático, duas novas pseudo instruções serão criadas: *BEGINMACRO* e *ENDMACRO*. A primeira é usada após uma label e indica o início das instruções de uma macro, que pode ou não ter um parâmetro. Já a segunda aponta o fim de uma macro.

2 – Implementação

Abaixo seguem a estrutura de dados utilizadas para a implementação do montador:

2.1 – Estruturas de dados:

```
#define TABLE_SIZE 200
#define INSTRUCTION_SIZE 50
#define MACRO_SIZE 50

typedef struct Macro {
    char macroName[INSTRUCTION_SIZE]; // Nome da macro
    char parameter[INSTRUCTION_SIZE]; // Parâmetro da macro
    int begin; // Começo no vetor de instruções de macro
    int end; // Final no vetor de instruções de macro
    int macroSize; // Tamanho total da macro: end - begin
} Macro;
```

```
typedef struct Expander {
    char macroInstructions[TABLE_SIZE][INSTRUCTION_SIZE]; // Armazena instruções de
                                                           // uma Macro
    Macro macro[MACRO_SIZE]; // Armazena informações das macros
    int numberOfMacros; // Numero de macros
} Expander;
```

2.2 – Funções:

```
void createExpander(int argc, char *argv[], Expander *expander); // Inicializa montador
com argumentos passados por linha de comando
```

```
int isLabel(char aux[]); // Verifica se string é label
```

```
int isBeginMacro(char aux[]); // Verifica se é inicio de declaração de macro
```

```
int isEndMacro(char aux[]); // Verifica se é fim de declaração de macro
```

```
int isBreakLine(char aux[]); // Verifica se é quebra de linha ou linha vazia
```

```
void removeComments(char aux[]); // Remove os comentários da linha
```

```
void getMacroInfo(Expander *expander, char input[]); // Pega as macros declaradas e armazena
```

```
int isMacroCall(Expander *expander, char aux[]); // Verifica se é chamada de macro
```

```
int getNumberOfOperands(char operation[]); // Retorna o número de operandos de uma operação
```

```
void replaceParameter(char macro[], char aux[], char toReplace[], char parameter[]);
// Troca o parâmetro passado pela macro
```

```
void expandMacros(Expander *expander, char input[], char output[]); // Expande as macros
                                                                    chamadas
```

```
void writeInstructions(Expander *expander, FILE *outputProgram, int macroId, int begin, int
end, char macroParameter[]); // Escreve as instruções das macros expandidas
```

3 – Decisões de projeto

De forma similar aos trabalhos anteriores, visando se ter uma maior modularidade no código, bem como o tornar mais fácil de se alterar, este foi dividido em varias funções menores que executam tarefas específicas. Assim a função principal do programa tem conteúdo reduzido.

Foi implementado então um tipo abstrato de dados chamado Expander, permitindo, caso necessário, a adição de novos campos ao expansor construído neste trabalho, mantendo as estruturas de dados em uma forma organizada e de fácil entendimento.

Além disto o código foi dividido em arquivos **.c** e **.h**:

main.c

Arquivo que contém a função *main* do programa. A partir deste arquivo são chamadas as

principais funções que fazem o montador construído funcionar.

expander.c

Arquivo que contém funções chamadas pela *main* além de funções auxiliares que realizam operações específicas, rodando o expensor.

expander.h

Arquivo que contém a estrutura de dados construída para o montador e o cabeçalho das funções utilizadas no arquivo *expander.c*.

4 – Compilação e Execução

4.1 – Compilação

O programa pode ser compilado utilizando *Makefile* através do comando *make* ou alternativamente pode ser compilando utilizando-se:

```
gcc -Wall main.c expander.c -o expensor
```

4.2 – Execução

Para executar o programa compilado é necessário utilizar o comando:

```
./expensor <entrada.amv> <saida.amv>
```

Onde:

entrada.amv – Localização do arquivo que contém o programa de entrada podendo conter macros

saida.mv – Localização do arquivo para o qual será escrito o programa com as macros chamadas expandidas. Caso não exista o arquivo será criado.

4.3 – Arquivo de entrada

Um simples exemplo do arquivo de entrada de instruções em linguagem de alto nível pode ser visto abaixo. O programa a seguir é um loop do qual se sai apenas se o número digitado pelo usuário for “0”.

```
READ R0
READ R1
PRINT R0
PRINT R1
ADD R0 R1
COPY R2 R0
PRINT R2
PRINT: BEGINMACRO REG
WRITE REG
ENDMACRO
END
```

4.4 – Saída

Como não há o argumento para “verboso” neste trabalho, nada é impresso na tela e o resultado pode ser visto no arquivo de saída. Este arquivo de saída pode então ser usado como arquivo de entrada para o montador. O arquivo de saída do montador por sua vez pode ser usado como arquivo de entrada do emulador para executar o programa escrito. O programa acima com as macros expandidas seria então:

```
READ R0
READ R1
WRITE R0
WRITE R1
ADD R0 R1
COPY R2 R0
WRITE R2
END
```

As instruções PRINT foram substituídas pelas instruções correspondentes dentro da definição da macro e os parâmetros foram passados, escrevendo as instruções corretamente.

5 – Testes

Uma série de testes foram realizados de modo a garantir que as instruções seriam executadas de modo correto e sem erros. Sendo eles:

macroafterlabel.amv – Lê 4 números e caso o 1º deles seja negativo, imprime o primeiro número. Caso seja zero, imprime o segundo número. Caso seja positivo imprime o números em ordem somado aos anteriores.

macroinsidemacro.amv – Lê 4 números e imprime os números em ordem somados aos anteriores.

macrowithoutparameter.amv – Lê 2 números e imprime o primeiro, o segundo e a soma de ambos.

macrowithparameter.amv – Lê 2 números e imprime o primeiro, o segundo e a soma de ambos.

median.amv – Calcula a mediana de 7 números, utilizando macros.

Além dos testes “padrão”, a fim de cumprir o proposto pela especificação foram desenvolvidos mais dois programas sendo eles:

factorial.amv – Calcula o fatorial de um dado número.

fibonacci.amv – Retorna o número na série de Fibonacci correspondente.

gdc.amv – Calcula o máximo divisor comum entre 2 números.

Pode-se então executar o expansor para *factorial.amv*, através do comando:

```
./expansor factorial.amv factorialExpanded.amv
```

Após isto feito podemos utilizar *factorialExpanded.amv* como entrada para o montador, como visto na imagem abaixo.

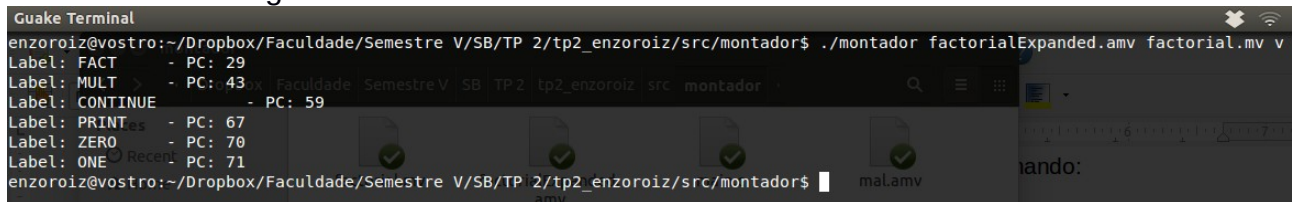


Figura 1 – Execução do montador em modo verboso

É gerado então o código de máquina, que pode ser utilizado como entrada para o emulador, como feito no exemplo abaixo.

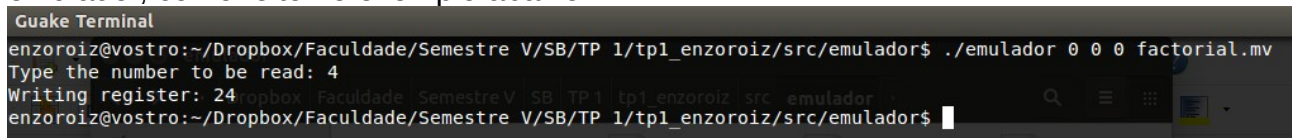


Figura 2 – Execução do emulador

6 – Conclusão

Para este trabalho foi construído um expansor que verifica as macros definidas em um programa de entrada e as expande quando são chamadas, gerando um arquivo de saída. Com isto é possível a partir de agora escrever os programas necessários de uma forma mais intuitiva, utilizando-se de modularização quando necessário através das macros.

Neste trabalho foram encontradas algumas dificuldades, entre elas a implementação da chamada de macros dentro de macros e o uso de macros após labels. Apesar destas dúvidas terem surgido, todas foram sanadas através do Moodle da matéria.

O expansor de macros passou por uma série de testes padrões e programas exigidos pela especificação e para todos teve a saída esperada, mostrando atender as especificações deste trabalho.