Trabalho Prático 2

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.

Para o segundo trabalho prático objetiva-se a construção de um montador que permita fazer a tradução de um programa em linguagem mais alto nível, similar a assembly, para linguagem de máquina. A saída do segundo trabalho prático, pode então ser utilizada como entrada para o emulador criado no primeiro trabalho prático.

Através de um arquivo, passado como parâmetro para o programa, o programa faz a tradução da linguagem similar a assembly para linguagem de máquina. Esta tradução ocorre em dois passos: No primeiro passo é criada uma tabela de símbolos com as labels existentes e no segundo passo é feita a tradução propriamente dita utilizando-se da tabela de símbolos criada no primeiro passo.

Além das instruções já implementadas no primeiro trabalho prático, duas pseudo-instruções, servem apenas para o montador, foram implementadas sendo elas: *WORD* e *END.* A primeira aloca uma posição de memória cujo valor é o inteiro após a instrução e a segunda determina quando o programa deve terminar.

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 100

typedef struct Mounter {
   int symbolPC[TABLE_SIZE]; // Armazena o Program Counter
   char symbolLabel[TABLE_SIZE][INSTRUCTION_SIZE]; // Armazena a label
   int verbose; // Armazena se verboso
} Mounter;
```

2.2 – Funções:

void createMounter(int argc, char *argv[], Mounter *mounter); *// Inicializa montador com argumentos passados por linha de comando*

void createSymbolTable(Mounter *mounter, char input[]); *// 1º passo: Cria tabela de símbolos*

int isLabel(char aux[]); *// Verifica se instrução é ou não label*

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

int getNumberOfOperands(char aux[]); *// Retorna o número de operandos da instrução passada por parâmetro*

void translateProgram(Mounter *mounter, char input[], char output[]); *// 2º passo: Tradução propriamente dita*

int getLabelPC(Mounter *mounter, char label[]); *// Retorna o PC da label passada por parâmetro*

void removeComments(char aux[]); *// Remove os comentários se existentes*

int getRegister(char reg[]); *// Retorna o registrador em forma de número*

3 – Decisões de projeto

De forma similar ao primeiro trabalho implementado, 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 Mounter, permitindo, caso necessário, a adição de novos campos ao montador 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.

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

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

4 – Compilação e Execução

4.1 – Compilação

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

```
gcc -Wall main.c mounter.c -o montador
```

4.2 – Execução

Para executar o programa compilado é necessário utilizar o comando:

```
./montador <entrada.amv> <saida.mv> [s|v]
```

Onde:

entrada.amv – Localização do arquivo que contém o programa em linguagem de mais alto nível, similar a assembly
saida.mv – Localização do arquivo para o qual será escrito o programa traduzido, em linguagem de máquina, caso não exista o arquivo será criado
[s|v] – Parâmetro opcional, escolhe se programa executará em modo verboso

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”.

```
START: READ R0
COPY R1 R0
JZ FINISH
JNZ START
FINISH: WRITE R0
HALT
```

4.4 – Saída

Caso o argumento para “verboso” seja *s* ou o argumento seja omitido, o programa é executado sem que seja exibida na tela a tabela de símbolo construída para o montador. Caso o programa seja executado em modo verboso é possível acompanhar ver a tabela de símbolos como mostrado a seguir:

Label: START - PC: 1
Label: FINISH - PC: 10

5 – Testes

Uma série de testes foram realizados de modo a garantir que as instruções seriam executadas de modo correto e sem erros. Assim todas as instruções no leque de instruções foram testadas.

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

median.amv – Calcula a mediana de 7 números
fibonacci.amv – Retorna o n-ésimo número da série de Fibonacci
division.amv – Calcula o quociente e o resto da divisão entre dois números
exponentiation.amv – Calcula o resultado da exponenciação entre 2 números.

Abaixo pode-se ver a execução do montador para a série de Fibonacci.

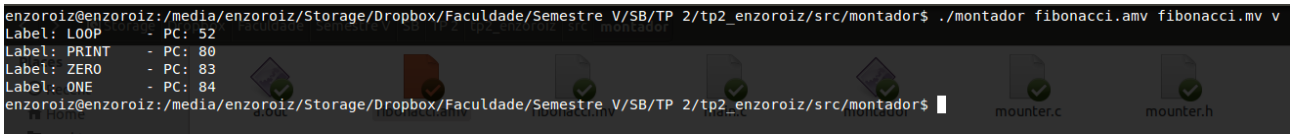

Figura 1 – Execução em modo verboso

6 – Conclusão

Para este trabalho foi construído um montador de dois passos que traduz um programa de uma linguagem mais alto nível, similar a assembly para linguagem de máquina. Com isto é possível a partir de agora escrever os programas necessários de uma forma mais intuitiva ao invés de utilizar apenas números, dificultando a compreensão.

Foram encontradas algumas dificuldades na implementação deste trabalho como por exemplo o entendimento da execução da pseudo-instrução WORD, porém todas as dúvidas foram sanadas através do Moodle da matéria.

Além disto, através deste trabalho prático foi possível reforçar o entendimento sobre como a máquina virtual implementada no primeiro trabalho prático deveria funcionar e foi possível consertar alguns problemas encontrados durante a execução deste.

A geração da tabela de símbolos, bem como a tradução para todas as instruções e pseudo-instruções foram testadas e a máquina atende às especificações propostas.