TRABALHO PRÁTICO 3: Expansor de Macros

Felipe Moraes Gomes - felipemoraes@dcc.ufmg.br

1 Introdução e Definição do Trabalho

Este trabalho descreve a implementação e operação de um expansor de macros para uma linguagem assembly hipotética, baseada no conjunto de instruções do RISC. O programa recebe como entrada um arquivo de texto, contendo código, e identifica as macros nele, efetuando sua expansão.

O restante deste documento está organizado da seguinte forma: A 2ª seção trata da implementação do expansor e de sua organização no código; A 3ª seção resume o formato de execução, entrada e saída do programa; A 4ª seção contém os testes realizados; A 5ª seção conclui o trabalho. Após isso, é colocado um apêndice contendo uma listagem dos arquivos do projeto.

2 Implementação e Organização

O expansor abre o arquivo de código especificado e efetua sua expansão em dois passos. Na primeira leitura, todos os macros são identificados e armazenados. Em seguida, uma segunda leitura é realizada substituindo as macros identificadas por uma instanciação de seu código.

O código foi implementado através de funções e procedimentos com uma estrutura de dados para auxiliar o armazenado e expansão das macros, delimitadas a seguir.

2.1 Dados e Variáveis

- *vector<struct> Macro*: Armazena as macros obtidas no primeiro passo.
- *struct<Macro>*: Armazena o tipo de cada macro nos campos do registro. Os tipos são *Label*, *Parameter* (se for uma instancia do parâmetro da macro), e *Body* (código identificado).

2.2 Procedimentos e Funções

- *int ExtractMacros(char* filename, Macros)*: Dado o nome do arquivo, extrai todas macros existente no programa de entrada e armazena em um conjunto de Macros retornando o número de macros encontradas.

- *void ExpandMacros(char* input, char* output, Macro* Macros, int num)*: Lê uma linha do arquivo de fonte, se for uma instrução imprime no arquivo de saida, se não procura pela macro preenchendo com parametros passados para ela.

- *char* SearchMacro(char *Label_macro, Macro *Macros, char *Parameter, int num)*: Dado um label, procura uma macro no conjunto de Macros, caso a macro tenha um parametro a função *ReplaceParameter* é chamada e em seguida a função retorna uma *string* contendo o corpo da macro.

- *char* ReplaceParameter(Macro macro, char *Parameter)*: Substitui na macro fornecida como entrada, o parâmetro, retornando uma *string* contendo o corpo da macro.

2.3 Fluxo de Execução

O programa inicialmente lê os parâmetros passados a ele, verificando seu formato. Caso correto, ele chama *ExpanderExpand*, que primeiro identifica as macros, e depois gera o arquivo de saída, em formato de código, já se tendo efetuado as expansões. Após retornar, o programa conclui sua execução.

3 Controle & IO

3.1 Execução e Compilação do Expansor

O programa pode ser compilado através do g++, pelo utilitário *make*, usando o makefile providenciado. Uma vez compilado, chamadas devem seguir o formato:

./bin/expansor input.amk output.amk

Onde *input* e *output* são os arquivos de código com e sem macros, respectivamente.

3.2 Formato dos Arquivos de Entrada e Saída

Cada linha dos programas em assembly devem seguir o formato:

[<label>:] <instrução> <operando1> <operando2> [; comentário]

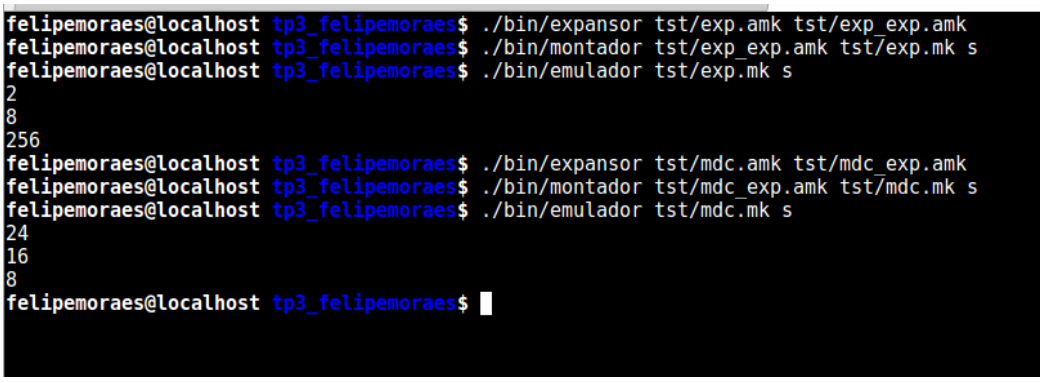

Figura 1: Compilação e execução dos testes

Onde *label* e *comentário* são opcionais, porém devendo ser devidamente delimitados (o sufixo ":" para o label e o prefixo ";" para o comentário). O tipo e quantidade dos operandos é dependente da instrução.

Macros são delimitados por *BEGINMACRO* e *ENDMACRO*, podendo haver um parâmetro opcional, permitindo substituição literal nas instanciações das macros. Não podem haver definições ou declarações de macros dentro de outras macros;

O arquivo de saída obtido obedece o mesmo formato do arquivo de entrada, excetuando a presença de macros, que sofreram substituição.

4 Testes

Vários programas foram testados, de forma a obter cobertura total do código. Os testes podem ser executados na MV tornando $PC = EndInicial = 0$ e $SP = 1000$. Os testes fizeram forte uso do expansor de macros, que promove grandes alterações sintáticas, visando aproximar uma linguagem mais alta.

- *Exponenciação (exp.amk):* Recebe dois parâmetros A e B, e imprime o resultado de A^B. Para tal, são computados os valores de A^{2^i} onde $i \in (1, 32)$, que, subsequentemente, são multiplicados uns com os outros até que a soma dos expoente dê B. O algoritmo é $O(log(B)log(A))$, pois cada produto é $O(log(A))$ e o número de produtos é $O(log(B))$.

- *MDC (mdc.amk):* Recebe dois números, e imprime o máximo divisor comum (MDC) deles. O cálculo é feito usando o algoritmo de MCD binário (uma extensão do algoritmo Euclidiano), e é $O(log^2(AB))$.

5 Conclusão

O expansor é eficaz em lidar com programas que fazem uso extensivo de macros, permitindo a construção de estruturas que auxiliem na elevação no nível da linguagem.

A execução do trabalho transcorreu sem maiores dificuldades, e os resultados obtidos correspondem ao esperado.

A Apêndice

A.1 Listagem de Arquivos

- Código Fonte:
  - *src/main.c:* Interpreta os parâmetros de entrada e controla o fluxo do programa (Seção 2.3).
  - *src/expander.c, src/expander.h:* Implementa o expansor (Seção 2.2).
  - *src/io.c, src/io.h:* Implementa instruções de entrada e saida.
  - *src/Makefile:* Makefile para facilitar a compilação do programa (Seção 3.1).
- Testes: Para cada teste, existe um *.sbasm* (o programa com macros), um *.sbint* (o código intermediário, sem macros) e um *.sbexe* (o executável que roda na MV).
  - *tst/exp.amk*: (Seção 4 item 1).
  - *tst/mdc.amk*: (Seção 4 item 2).