TRABALHO PRÁTICO 2: Montador

Felipe Moraes Gomes - felipemoraes@dcc.ufmg.br

1 Introdução e Definição do Trabalho

Este trabalho descreve a implementação e operação de um montador para uma linguagem assembly hipotética, baseada no conjunto de instruções do RISC. A máquina virtual recebe como entrada um arquivo de texto, contendo o código a ser montado, e alguns parâmetros de execução, que ele então usa para construir o executável.

O restante deste documento está organizado da seguinte forma: A 2ª seção trata da implementação do montador 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 montador abre o arquivo de código especificado e monta o executável em dois passes. Primeiramente, o arquivo é lido com o intuito de criar a tabela de símbolos. Em seguida, o arquivo é relido e traduzido, gerando o executável.

A verificação sintática foi implementado para auxiliar na depuração dos programas de teste desenvolvidos. O montador é representado por procedimentos do algoritmo principal, cujos componentes principais são delineadas a seguir.

2.1 Dados e Variáveis

Estruturas de Dados define várias estruturas de dados,

- *TipoLista symbol_table*: Tabela de símbolos que associa um label (string) com o ILC referente à aquele local.
- *TipoLista opcode_table*: Tabela de instruções que associa a string que identifica a instrução no assembly com seu respectivo índice no código de máquina. As pseudo-instruções *WORD* e *END* foram atribuídos os valores 22 e 23, respectivamente (muito embora estes valores são usados somente internamente, ausentando-se do executável). As demais instruções são codificadas conforme a especificação.

- *TipoLIsta size_table*: Tabela de instruções que associa o índice da instrução com seu tamanho.

2.2 Procedimentos e Funções

- *char* readline()*: Lê uma linha do arquivo fonte e a retorna.
- *char* next_word*: Extrai a próxima palavra de uma linha, apagando ela na linha original, retorna nulo caso linha vazia, ou somente comentários.
- *void Decode()*: Recebe uma linha do arquivo de código fonte, e a decodifica, retornando o índice da instrução (usando *opcode_table*), e, se for o caso, o label da linha e o número e valor dos operandos. Caso seja encontrada uma instrução desconhecida ele segue para frente.
- *void Primeiro_Passo()*: Abre o arquivo fonte, e efetua a primeira passada, construindo a tabela de símbolos. Para tal, *readline()* é chamado continuamente até que *END* seja encontrado. O ILC é simulado a cada instrução e, se na linha houver um label, é associado a ele pela lista *symbol_table*.
- *void Segundo_Passo()*: Efetua a segunda passada, fazendo a tradução e escrevendo o executável. Para tal, *Decode* é chamado continuamente até que *END* seja encontrado. Cada instrução decodificada é traduzida e colocado na saída, substituindo os labels das referências de memória pelo PC associado (*symbol_table*), observando que no caso da pseudo-instrução *WORD*, somente o operando é usado.
- Os restantes das funções, são de descrição desnecessária, uma vez que é usada continuamente em muitas disciplinas do curso.

2.3 Fluxo de Execução

O programa inicialmente lê os parametros passados a ele, verificando seu formato. Caso correto, ele constrói a tabela de instruções (*opcode_table*). Em seguida, o arquivo contendo o código fonte é aberto, e sua tabela de símbolos é construída (primeiro passe). Finalmente, *Segundo_Passo()* é chamado, que efetua a tradução, e monta o executável. Ao retornar, o programa imprime a tabela de símbolos (se o mesmo foi especificado) e conclui sua execução.

3 Controle & IO

3.1 Execução e Compilação do Simulador

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

./montador input.amk output.mk 's'| 'v'

Onde 's' especifica que a saída deve ser simples (somente se for detectado um erro de sintaxe), e 'v' especifica que o programa deve também imprimir a tabela de símbolos. *input* é o nome do arquivo de entrada que contém o código assembly a ser traduzido, e *output* é o nome do arquivo executável gerado pelo montador.

3.2 Formato dos Arquivos de Entrada e Saída

Cada linha dos programas em assembly devem seguir o formato:

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

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

Os programas gerados são codificados em um arquivo de texto, onde cada linha contém exatamente um *WORD*, que representa um dado ou uma instrução, segundo a codificação delimitada na especificação.

4 Testes

Múltiplos programas foram montados, de forma a obter cobertura total do código. Os testes podem ser executados no emulador. As tabelas de símbolos obtidas durante a tradução de cada programa são ilustradas na Fig. 1 e 2.

- *Divisão força bruta (divisao.amk):* Faz a divisão força bruta de dois números (passados como parâmetros), e imprime o quociente e o resto obtido. O tratamento de valores negativos foi feito de tal forma que a soma do resto com o produto do quociente com o divisor sempre desse o numerador (com $0 \leq resto < |Divisor|$).

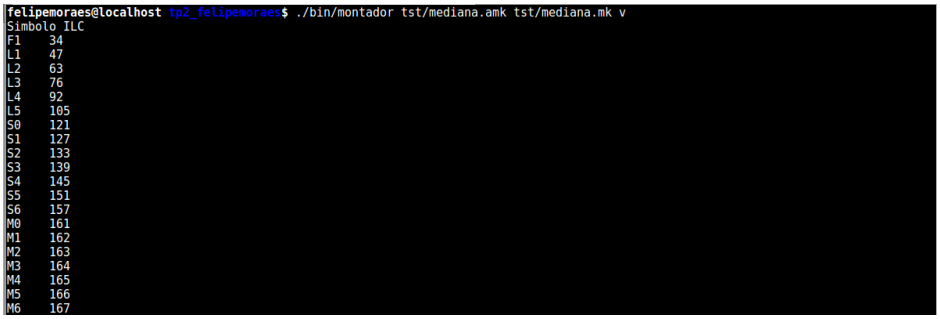

Figura 1: Montagem dos testes - Teste Mediana

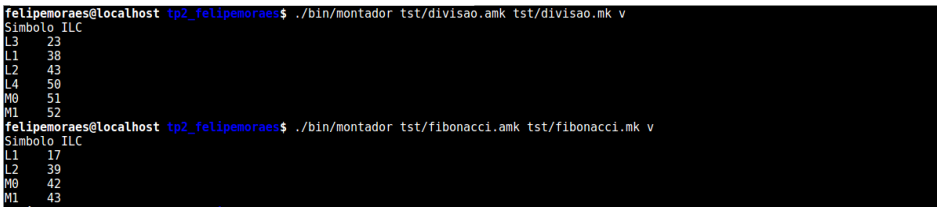

Figura 2: Montagem dos testes - Teste Divisão e Fibonacci

- *Fibonacci (fibonacci.amk)*: Imprime o enésimo número da sequência fibonacci, onde N é passado como parâmetro.
- *Mediana de 7 (mediana.amk)*: Dado um conjunto de 7 números (lidos do terminal), imprime sua mediana.

5 Conclusão

O montador é eficaz em lidar com programas escritos neste conjunto de instruções, podendo ser usado não só para traduzi-los mas também para adequadamente depurar erros de sintaxe.

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/assembler.c, src/assembler.h:* Implementa o montador (Seção 2.1 e 2.2).
  - *src/io.c, src/io.h:* Implementa funções de entrada e saida.
  - *src/lista.c, src/lista.h:* Implementa a TAD lista encadeada.
  - *src/Makefile:* Makefile para facilitar a compilação do programa (Seção 3.1).

- Testes:
  - *tst/divisao.amk; tst/divisao.mk:* (Seção 4 item 1).
  - *tst/fibonacci.amk; tst/fibonacci.mk:* (Seção 4 item 2).
  - *tst/mediana.amk; tst/mediana.mk:* (Seção 4 item 3).