MONTADOR
RELATÓRIO

Álissom Vieira da Cunha

00216123

Luiz Eduardo Pereira

0021619

Instituto Federal de Minas Gerais, Formiga, MG.

INTRODUÇÃO

Este trabalho tem como objetivo a implementação de um montador para o processador teórico MIPS-IF32, inspirado em um MIPS de 32 bits.

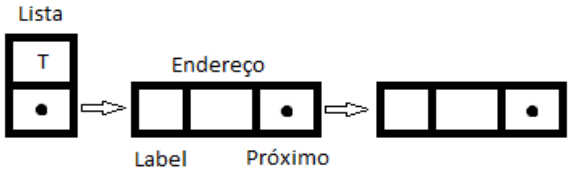
IMPLEMENTAÇÃO

Este trabalho foi desenvolvido na linguagem C e foi dividido em 3 estruturas: Arquivo.c, Lista.c e Montador.c

Arquivo.c

A parte de Arquivo faz simplesmente as operações básicas com arquivo texto, abertura, fechamento, leitura e escrita.

Lista.c



A Lista Encadeada é usada para armazenar os Labels do código. Ela armazena o nome do label (char*), um endereço (int) que guarda a posição da memória usada pelo label e um ponteiro para o proximo nodo.

A implementação de lista para dados e lista para instruções utilizam o mesmo tipo de estrutura (lista), mas são armazenadas em variáveis diferentes.

Montador.c

O montador possui 4 variáveis globais:

lista_data, lista_text (Lista):
Guarda o nome dos labels e o endereço correspondente.

mem_data, mem_text (int):
Guarda a quantidade em bytes a ser utilizado pelo programa. No final, essas variáveis são convertidas em words (4 bytes);

Divisão

O montador é dividido em 3 partes principais: Inicializador, Construtor e Funções Uteis:

Inicializador

O inicializador possui funções que irão construir as instruções.

A função le_data faz a leitura do arquivo procurando por labels e diretivas, e fazendo suas respectivas traduções. Para não precisar ler a parte de dados duas vezes, sendo que na primeira seria necessário contar a memória, e na segunda fazer a tradução, optamos por fazer a tradução na primeira leitura, e armazena-lo em um arquivo temporário. Terminando a leitura, já é possível saber o quanto de memoria a parte de dados necessitara, então o arquivo de saída recebe o ".data <inteiro>" e a tradução armazenada no arquivo temporário é transferida para o arquivo de saída.

A função le_label faz a leitura do arquivo procurando por labels dentro de .text e armazenando seus nomes e endereços na lista_text.

A função le_instrucao faz a leitura do arquivo na parte do .text, dessa vez, ignorando todos os labels. Se encontra uma instrução valida, verifica se é uma pseudo-instrução ou uma instrução comum, desse modo, o programa entra na parte do Construtor.

Construtor

O construtor recebe uma instrução e o transforma em binário e depois hexadecimal.

A função verifica_instrucao encontra o tipo de instrução e a traduz para binário. Essa função utiliza a bloco_instrucao e bloco_especial que pegam os outros parâmetros (registradores, imediatos e labels) para serem traduzidos.

A função verifica_pseudo funciona de maneira similar à verifica_instrucao, a principal diferença, e que ela devolve duas instruções (nem sempre). Na parte do inicializador le_instrução a instrução é quebrada em duas (se necessário) para formar

duas words. Optamos por replicar as instruções simples na função virifica_pseudo, pois haveria um pouco de dificuldade em reaproveitar a função verifica_instrução.

A função verifica_diretiva encontra a diretiva usada pela parte de dados, e a traduz de acordo com a especificidade de cada diretiva.

A função verifica_registrador retorna o binário de cada registrador.

Funções Úteis

Funções Úteis são várias funções que o inicializador e o construtor utilizam para determinadas tarefas, como:

Pegar parâmetros;

Conversão de bases;

Formatação de instrução para um formato legível (removendo espaços, tabulações, etc.).

Dentre outros;

VALIDAÇÃO

Para fazer os testes, comparamos resultados de determinados arquivos com outros colegas de turma.

PENDÊNCIAS

Não deixamos nenhuma pendência conhecida no trabalho.

CONCLUSÃO

Este trabalho ajuda no entendimento de como os programas são feitos em baixo nível. Também nos mostra que devemos ficar atento ao jeito de programar, que uma instrução em C, por exemplo, pode ser subdividida em várias instruções em assembly, tendo esse conhecimento, podemos conseguir fazer códigos mais otimizados.

Dificuldades encontradas: implementação de pseudo-instruções e diretivas. Devemos agradecimentos a vários alunos da disciplina que compartilharam suas ideias para que todos conseguíssemos chegar ao resultado final.