UNIVERSIDADE ESTADUAL DE MARINGÁ
DEPARTAMENTO DE INFORMÁTICA
Curso: Bacharelado em Informática
Professor: Nilton Luiz Queiroz Junior
Arquitetura e Organização de Computadores

Discentes:

Raphael Franco de Lima	RA: 98336
Gustavo Ast Mariano	RA: 99843
Aitor Vieira Vieira	RA: 98483

Relatório Simulador de Arquitetura

Maringá

20 de julho de 2018

Introdução

O presente relatório tem por intuito descrever a construção de um simulador para arquitetura desenvolvido como trabalho para a disciplina 5178 - Arquitetura e Organização de Computadores. Para tal desenvolvimento o principal elemento contribuinte fora a utilização do software gerador de análise léxica denominado "flex".

A análise léxica é o processo de analisar a entrada de linhas de caracteres e produzir uma sequência de símbolos léxicos que podem ser manipulados mais facilmente no futuro. Este processo é, comumente, a primeira etapa no processo de compilação e prepara o código para a análise sintática.

Além disso foi estudada a utilização de um *parser,* porém devido a baixa complexidade do projeto esta ideia fora descartada.

Desenvolvimento

Inicialmente foi feita uma análise da problemática e identificado que o ponto crítico do simulador seria identificar os comandos presentes em cada instrução, sendo assim necessária uma grande manipulação de strings. Foi visto então que a utilização de recursos de análise léxica seriam uma boa forma de sanar tal problema.

A ferramenta "flex" fora utilizada devido a grande confiabilidade que a mesma possui e também em detrimento do vasto conteúdo técnico disponível acerca de sua utilização. O seu funcionamento consiste basicamente em encontrar nos caracteres de entrada padrões definidos anteriormente. Esses padrões podem valer da utilização de expressões regulares ou literais.

A sintaxe geral de um programa flex consiste em três seções delimitadas pelo símbolo %. A primeira seção contém declarações gerais (includes, variáveis globais, etc) e configurações adicionais. A segunda seção é composta pelos padrões a serem encontrados e a respectiva ação a ser executada. Por fim a terceira seção contém código em C responsável por chamar a subrotina do analisador léxico entre outras ações definidas pelo programador.

Essa divisão pode ser constatada no simulador desenvolvido como se segue.

```
%{
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
#define pollingDelay 5000
char *ptr;
long val;
int tokens[5], i = 0, pc = 0;

%}
```

Figura 1. Primeira seção do código fonte

Na primeira seção foram declaradas algumas variáveis globais, juntamente de algumas bibliotecas necessárias para o funcionamento do programa.

```
%%
"add" {tokens[i] = 0; i++;}
"addi" {tokens[i] = 1; i++;}
"sub" {tokens[i] = 2; i++;}
"subi" {tokens[i] = 3; i++;}
"blt" {tokens[i] = 4; i++;}
"bgt" {tokens[i] = 5; i++;}
"beq" {tokens[i] = 6; i++;}
"lw" {tokens[i] = 7; i++;}
"sw" {tokens[i] = 8; i++;}
"mov" {tokens[i] = 9; i++;}
"r0" {tokens[i] = 10; i++;}
"r1" {tokens[i] = 11; i++;}
"r2" {tokens[i] = 12; i++;}
"r3" {tokens[i] = 13; i++;}
"r4" {tokens[i] = 14; i++;}
"r5" {tokens[i] = 15; i++;}
"r6" {tokens[i] = 16; i++;}
"r7" {tokens[i] = 17; i++;}
"r8" {tokens[i] = 18; i++;}
[^r][0-9]+ {val = strtol(yytext, &ptr, 10); tokens[i] = val; i++;}
. {}
%%
```

Figura 2. Segunda seção do código fonte

A segunda seção, e mais importante, contém os padrões literais que queremos identificar. Como a análise léxica é feita de maneira sequencial na instrução de entrada a estratégia utilizada para o simulador foi a de cada comando possível de aparecer em uma instrução ter um valor inteiro relacionado, e este seria adicionado em uma variável que armazenaria todas as ocorrências em uma instrução.

As duas últimas linhas de código desta seção mereceriam atenção especial. A penúltima linha seria composta por uma expressão regular responsável por identificar os valor de imediato na instrução e posteriormente transformar tal valor (cujo inicialmente é uma string) em um inteiro.

Já a última linha também corresponde a uma expressão regular composta unicamente pelo caractere ponto ‘.’ cujo possui a função de encontrar qualquer outro caractere ou padrão não especificado e posteriormente não realizar ação alguma. Esta estratégia foi utilizada pois do contrário toda ocorrência diferente dos padrões definidos seriam expostos através de um comando de saída (comportamento nativo do programa flex).

Por fim pode ser apresentado um trecho da terceira seção

```c
int main(void){
    void (*f[])(int *, int *) = {add, addi, sub, subi, blt, bgt, beq, lw, sw, mov};
    char instr[100][255], ir[1][255]; //instruções
    int regist[9], memo[100], j = 0;
    regist[0] = 0;

    FILE *fp;
    YY_BUFFER_STATE bp;
    fp = fopen("instrucoes.txt", "r");
    while(fgets(instr[j], 255, fp) != NULL)
        j++;
    fclose(fp);

    while(pc < j){
        strcpy(ir[0], instr[pc]);
        bp = yy_scan_string(ir[0]);
        yy_switch_to_buffer(bp);
        yylex();
        yy_flush_buffer(bp);
        pc++;
        f[tokens[0]](regist, memo);
        i = 0;
        printMemory(memo);
        printRegist(regist);
        printGenReg(ir);
        delay();
    }

    yy_delete_buffer(bp);
    return 0;
}
```

Figura 3. Terceira seção do código fonte

O trecho exibido acima suprime todas as funções responsáveis por realizar as operações definidas no conjunto de instruções haja vista que essas possuem baixa complexidade. Porém, vale pontuar o recurso utilizado para realizar a chamada de tais funções. O recurso referido fora criar um vetor de ponteiro para função, tal solução foi proposta haja vista já existir um valor numérico que identifica cada uma das operações, facilitando assim a sua chamada através da solução já citada (pode usar como analogia o funcionamento de uma tabela hash).

Além disso o trecho de código exibe comandos responsáveis por realizar a leitura do arquivo de instruções e passagem das mesmas para uma matriz de strings (o termo correto seria matriz de caracteres, porém para facilitar o entendimento o termo fora alterado). Essa estratégia foi pensada haja vista que instruções de salto exigem um meio para indexar instruções e avançar ou retroceder entre elas. Por fim códigos que controlam a rotina do analisador léxico são definidos juntamente de

instruções para imprimir o estado de diversas variáveis solicitadas na descrição do projeto.

O arquivo de instruções deve ser no formato “.txt” e conter somente caracteres em minúsculo. Para execução do programa deve-se executar o seguinte conjunto de comandos.

flex simulador.c // Através deste comando o programa flex realiza a tradução do arquivo

gcc lex.yy.c -lfl // Comando para compilação do arquivo gerado no passo anterior e linkagem da biblioteca necessária

./a.out // Execução do simulador

Conclusão

O trabalho apresentou resultado satisfatório haja vista o sucesso na implementação do simulador comprovado através dos múltiplos testes feitos para diversos conjuntos de instrução. Nota-se que a ferramenta “flex” auxiliou fortemente na ação de reconhecer os padrões necessários e por consequência agilizar o desenvolvimento.