Минобрнауки России

Санкт-Петербургский государственный политехнический университет

Институт информационных технологий и управления

Кафедра «Информационные и управляющие системы»

К У Р С О В О Й П Р О Е К Т

Р а з р а б о т к а у ч е б н о й с и с т е м ы п р о г р а м м и р о в а н и я

В а р и а н т 7

Построение компилятора с ЯВУ с использование flex/bison

по дисциплине «Системы программирования»

Выполнили

студенты гр.5084/12 А.А.Лукашин

К.С.Шубин

Руководитель

доцент В.Я.Расторгуев

«__» _____ 2013 г.

Санкт-Петербург

2013

Оглавление

Задание

Разработать компилятор с языка высокого уровня (ЯВУ) с использованием технологий flex/bison для получения эквивалента исходного текста на ассемблере.

Вариант №7:

Код на языке PL1

```
EX07: PROC OPTIONS (MAIN);

            DCL A BIT (3) INIT ( 10B );

            DCL B BIT (3) INIT ( 101B );

            DCL C BIT (16);

            C = SUBSTR((B !! A),2,3);

END EX07;
```

Технология

В рамках данного этапа предполагается использование FLEX(fast lexical analyzer) – генератора лексических анализаторов. Для использования flex необходимо задать правила выделения лексем. Далее flex принимает на вход исходный текст(в нашем случае это код на языке PL1, указанный в задании), на выход выдается лексический анализатор (функция на языке C), при этом гарантируется, что построенный анализатор будет не хуже, написанного вручную. Данный анализатор имеет функцию для получения следующего токена из исходного текста.

Полученный анализатор используется дальше синтаксическим анализатором bison, который находит нетерминальные элементы и генерирует выход согласно заданной грамматике.

Таким образом для выполнения поставленного задания необходимо решить следующие задачи:

1) Описать правила выделения лексем для flex
2) Сгенерировать лексический анализатор
3) Описать синтаксические правила для bison
4) Проверить правильность работы компилятора на исходном примере

Примечание:

Так как данный этап курсовой работы повторяет задание 1го этапа, изменения в синтаксических правилах и грамматике языка PL1 с учетом задания были опущены.

Эквивалент на Ассемблере

После компиляции с языка PL1 на Ассемблер должен получиться следующий код:

```
EX07     START 0            Начало программы
         BALR  RBASE,0      Загрузка регистра базы
         USING *,RBASE      Назначить регистр базой
         LH    3,B          Загрузка переменной B в регистр
         LH    4,A          Загрузка переменной A в регистр
         OR    3,4          Логическое «ИЛИ» регистров
         SRL   4,3          Сдвиг операнда вправо
         SLL   3,2          Сдвиг операнда влево
         LH    4,TMP        Загрузка маски в регистр
         NR    3,4          Логическое «И» регистров
         STH   3,C          Формирование результата
         BCR   15,RVIX      Выход из программы
A        DS    0H           Выравнивание адреса
         DC    BL2'10'      Инициализация переменной
B        DS    0H           Выравнивание адреса
         DC    BL2'101'     Инициализация переменной
C        DS    0H           Выравнивание адреса
         DS    BL2          Объявление без инициализации
TMP      DS    0H           Выравнивание адреса
         DC    BL2'111'     Инициализация переменной
RBASE    EQU   5
RVIX     EQU   14
         END   EX07         Конец программы
```

Первые 3 строки программы – пролог.

Следующие 4 строки относятся к операции конкатенации. Переменная B записывается в 3 регистр, переменная A – в 4 регистр. Далее происходит сдвиг содержимого регистра 4 на 3 разряда вправо (SRL). Затем регистровая операция ИЛИ (OR) между 3 и 4 регистрами, результат записывается в 3 регистр. Для хранения переменных используется 16-разрядная модель данных (полуслово). Поэтому в качестве команды загрузки переменной в регистр используется LH.

Следующие 3 строки отвечают за операцию взятия подстроки. Происходит сдвиг влево содержимого 3 регистра на 2 разряда (второй параметр операции substr). Затем в 4 регистр загружается маска tmp. Первые 3 разряда маски (третий параметр substr) – единицы, остальное – нули. Затем происходит регистровая операции И между 3 и 4 регистрами (NR).

В следующей строке происходит запись содержимого из регистра в память.

Далее идёт объявление переменных. BL2 – значение переменной в логическом виде (выделяется 16 разрядов). Команда DS 0H служит для выравнивания адреса на границу полуслова. Параметр 0 позволяет осуществить выравнивания без выделения памяти.

Последние 3 строки программы – эпилог.

Исходный код

В данном разделе представлен исходный код приложения на языке C.

Лексический анализатор

```
%%
":"                    {
//                         printf("\n***lex: term=%s;\n", yytext);
                       return ':';
                       }
"("                    {
//                         printf("\n***lex: term=%s;\n", yytext);
                       return '(';
                       }
")"                    {
//                         printf("\n***lex: term=%s;\n", yytext);
                       return ')';
                       }
";"                    {
//                         printf("\n***lex: term=%s;\n", yytext);
                       return ';';
                       }
"="                    {
//                         printf("\n***lex: term=%s;\n", yytext);
                       return '=';
                       }

[+-]                   {
//                         printf("\n***lex: term=%s;\n", yytext);
                       yylval=strdup(yytext); return ZNK;
                       }
"!!"                   {
//                         printf("\n***lex: term=%s;\n", yytext);
                       yylval=strdup(yytext); return CONCAT;
                       }
"'"[0-9]*"'B"        {
```

```
//                        printf("\n***lex: term=%s;\n", yytext);
                        int ind = 0;
                        int i = 0;
                        char str[255];
                        for (; i < strlen(yytext); i++) {
                         if ((yytext[i] != '\'') && (yytext[i] != 'B')) {
                             str[ind++] = yytext[i];
                         }
                        }
                        str[ind] = 0x00;
                        yylval=strdup(str);
                        return VAL;
                      }
[a-zA-Z][a-zA-Z0-9]* {
//                        printf("\n***lex: term=%s;\n", yytext);
                        if (!memcmp(yytext,"proc", 4)) return PROC;
                        if (!memcmp(yytext,"options", 7)) return OPTIONS;
                        if (!memcmp(yytext,"main", 4)) return MAIN;
                        if (!memcmp(yytext,"end", 3)) return END;
                        if (!memcmp(yytext,"dcl", 3)) return DCL;
                        if (!memcmp(yytext,"bin", 3)) return BIN;
                        if (!memcmp(yytext,"bit", 3)) return BIT;
                        if (!memcmp(yytext,"fixed", 5)) return FIXED;
                        if (!memcmp(yytext,"init", 4)) return INIT;
                        if (!memcmp(yytext,"substr", 6)) return SUBSTR;
                        yylval=strdup(yytext); return IDENT;
                      }
[0-9][0-9]*          {
//                        printf("\n***lex: term=%s;\n", yytext);
                        yylval=strdup(yytext); return NUM;
                      }
[ \t\n]+             {
//                       printf("\n***lex: token is WightSpace\n");
                      }
```

```
%%
```

Синтаксический анализатор

1) Описание функций реализующих синтаксические правила

```
void pro();
void odi(char *tpe, char *rzr, char *lit);
void odr(char *tpe, char *rzr);
void opr(char *pr_name);
int  oen(char *pr_name);
int  opa(char *ipe);
int  opl(char *ipe);
void avi_lit(char *lit);
int  avi_ipe(char *ipe);
void avi_avi_znk_lit(char *znk, char *lit);
int  avi_avi_znk_ipe(char *znk, char *ipe);
int my_substr (char *ipe1, char *ipe2, char *shift, char *mask);

%}
%debug
%verbose
%token IDENT PROC OPTIONS MAIN END DCL BIN FIXED BIT NUM INIT VAL SUBSTR
%left ZNK CONCAT
%start pro
%%
pro: opr tel oen { pro(); }
     ;
tel: dec imp
     ;
dec: odc
    | dec odc
     ;
odc: odi
    | odr
     ;
odi:  DCL ipe BIN FIXED '(' rzr ')' INIT '(' lit ')' ';' { odi($2, $6, $10);
}
    | DCL ipe BIT '(' rzr ')' INIT '(' lib ')' ';' { odi($2, $5, $9); }
    ;
odr:  DCL ipe BIN FIXED '(' rzr ')' ';'                  { odr($2, $6); }
    | DCL ipe BIT '(' rzr ')' ';' { odr ($2, $5); }
     ;
ipe: IDENT {$$=$1;}
     ;
rzr: NUM {$$=$1;}
     ;
lit: NUM {$$=$1;}
     ;
shift: NUM {$$=$1;}
     ;
mask: NUM {$$=$1;}
     ;
lib: VAL {$$=$1;}
    ;
opr: IDENT ':' PROC OPTIONS '(' MAIN ')' ';'             { opr($1); }
     ;
oen: END IDENT ';'                                       { if ( oen($2) )
YYABORT; }
     ;

imp: opa
    | imp opa
    ;
```

```
opa: ipe '=' avi ';'                                        { if ( opl($1) )
YYABORT; }
    ;
avi: lit                                                    { avi_lit($1); }
    | ipe                                                   { if ( avi_ipe($1) )
YYABORT;}
    | avi ZNK lit                                           {
avi_avi_znk_lit($2, $3); }
    | avi ZNK ipe                                           { if (
avi_avi_znk_ipe($2, $3) ) YYABORT; }
    | SUBSTR '(' '(' ipe CONCAT ipe ')' shift  mask ')'   { if ( my_substr
($4, $6, $8, $9) ) YYABORT; }
    ;
%%
```

2) Описание реализаций указанных функций

- Пролог

```
 void opr(char *pr_name) {
               memset(&s1[0], ' ', 80);
               memcpy(&s1[0], pr_name, strlen(pr_name));
               memcpy(&s1[9], "START 0", 7);
               memcpy(&s1[30], "Programm start", 14);
               memcpy(&Prolog[0][0], &s1[0], 80);

               memset(&s1[0], ' ', 80);
               memcpy(&s1[9], "BALR  RBASE,0", 13);
               memcpy(&s1[30], "Base initialization", 19);
               memcpy(&Prolog[1][0], &s1[0], 80);


               memset(&s1[0], ' ', 80);
               memcpy(&s1[9], "USING *,RBASE", 13);
               memcpy(&s1[30], "Base declaration", 16);
               memcpy(&Prolog[2][0], &s1[0], 80);

               memcpy(&AssProgName[0], pr_name, strlen(pr_name));
}
```

- Объявление с инициализацией

```
void odi(char *ipe, char *rzr, char *lit) {

               memset(&s1[0], ' ', 80);
               memcpy(&s1[0], ipe, strlen(ipe));
               memcpy(&s1[9], "DS", 2);
               s1[15]='0';
               s1[16]='H';
               memcpy(&DclPart[pDclPart][0], &s1[0], 80);
               pDclPart++;

               memset(&s1[0], ' ', 80);
               memcpy(&s1[9], "DC", 2);
               char last_ind = 0;
               if(!memcmp(rzr, "31", 2)) {
                s1[15]='F';
                s1[16]='\'';
                last_ind = 17;
               } else if (!memcmp (rzr, "16", 2) || !memcmp (rzr, "3", 1)) {
                s1[15]='B';
```

```
                s1[16]='L';
                s1[17]='2';
                s1[18]='\'';
                last_ind = 19;
               } else {
                s1[15]='H';
                s1[16]='\'';
                last_ind = 17;
               }
               memcpy(&s1[last_ind], lit, strlen(lit));
               s1[last_ind+strlen(lit)]='\'';
               memcpy(&s1[30], "Variable declaration with initialization",
40);

               memcpy(&DclPart[pDclPart][0], &s1[0], 80);
               pDclPart++;
}
```

- Объявление без инициализации

```
void odr(char *ipe, char *rzr) {

                    memset(&s1[0], ' ', 80);
               memcpy(&s1[0], ipe, strlen(ipe));
               memcpy(&s1[9], "DS", 2);
               s1[15]='0';
               s1[16]='H';
               memcpy(&DclPart[pDclPart][0], &s1[0], 80);
               pDclPart++;

               memset(&s1[0], ' ', 80);
               memcpy(&s1[9], "DS", 2);
               if(!memcmp(rzr, "31", 2)) {
                s1[15]='F';
               } else if (!memcmp (rzr, "16", 2)) {
                s1[15]='B';
                s1[16]='L';
                s1[17]='2';
               } else {
                s1[15]='H';
               }
               memcpy(&s1[30], "Variable declaration without initialization",
43);

               memcpy(&DclPart[pDclPart][0], &s1[0], 80);
               pDclPart++;
}
```

- Функция substr

```
int my_substr (char *ipe1, char *ipe2, char *shift, char *mask) {
   if (IsDclName(ipe1, strlen(ipe1)) || IsDclName(ipe2, strlen(ipe2))){
       strcpy(&ErrorMessage[0], " invalid identificator ");
       strcat(&ErrorMessage[0], "in my_substr\n");
       yyerror(&ErrorMessage[0]);
       return 1;
   }
   memset(&s1[0], ' ', 80);
   memcpy(&s1[9], "LH", 2);
   memcpy(&s1[15], "3,", 2);
```

```
    memcpy(&s1[20], ipe1, strlen(ipe1));
    memcpy(&s1[30], "Zagruzka peremennoy v registr", 29);
    memcpy(&ImpPart[pImpPart][0], &s1[0], 80);
     pImpPart++;

    memset(&s1[0], ' ', 80);
    memcpy(&s1[9], "LH", 2);
    memcpy(&s1[15], "4,", 2);
    memcpy(&s1[20], ipe2, strlen(ipe2));
    memcpy(&s1[30], "Zagruzka peremennoy v registr", 29);
    memcpy(&ImpPart[pImpPart][0], &s1[0], 80);
     pImpPart++;

    memset(&s1[0], ' ', 80);
    memcpy(&s1[9], "SRL", 3);
    memcpy(&s1[15], "4,", 2);
    memcpy(&s1[20], "3", 1);
    memcpy(&s1[30], "Sdvig operanda vpravo", 21);
    memcpy(&ImpPart[pImpPart][0], &s1[0], 80);
     pImpPart++;

    memset(&s1[0], ' ', 80);
    memcpy(&s1[9], "OR", 2);
    memcpy(&s1[15], "3,", 2);
    memcpy(&s1[20], "4", 1);
    memcpy(&s1[30], "Logicheskoye ILI registrov", 26);
    memcpy(&ImpPart[pImpPart][0], &s1[0], 80);
     pImpPart++;

    memset(&s1[0], ' ', 80);
    memcpy(&s1[9], "SLL", 3);
    memcpy(&s1[15], "3,", 2);
    char realShift[15];
    snprintf(realShift, 15, "%d", atoi(shift)-1);
    memcpy(&s1[20], realShift, strlen(realShift));
    memcpy(&s1[30], "Sdvig operanda vlevo", 20);
    memcpy(&ImpPart[pImpPart][0], &s1[0], 80);
     pImpPart++;

    memset(&s1[0], ' ', 80);
    memcpy(&s1[9], "LH", 2);
    memcpy(&s1[15], "4,", 2);
    memcpy(&s1[20], "TMP", 3);
    memcpy(&s1[30], "Zagruzka maski v reg", 20);
    memcpy(&ImpPart[pImpPart][0], &s1[0], 80);
     pImpPart++;

    memset(&s1[0], ' ', 80);
    memcpy(&s1[9], "NR", 2);
    memcpy(&s1[15], "3,", 2);
    memcpy(&s1[20], "4", 1);
    memcpy(&s1[30], "Logicheskoye I registrov", 24);
    memcpy(&ImpPart[pImpPart][0], &s1[0], 80);
     pImpPart++;

    return 0;
}
```

- Эпилог

```
int oen(char *pr_name) {

               if (!memcmp(&Prolog[0][0], pr_name, strlen(pr_name))){

                memset(&s1[0], ' ', 80);
                memcpy(&s1[9], "END", 3);
                memcpy(&s1[15], pr_name, strlen(pr_name));
                memcpy(&s1[30], "Programm end", 12);
                memcpy(&Epilog[0], &s1[0], 80);

                memset(&s1[0], ' ', 80);
                memcpy(&s1[9], "BCR   15,  RVIX", 15);
                memcpy(&s1[30], "Return from programm", 20);

                memcpy(&ImpPart[pImpPart][0], &s1[0], 80);
                pImpPart++;

                return 0;
               }
               else {
                strcpy(&ErrorMessage[0], " invalid identificator ");
                strcat(&ErrorMessage[0], pr_name);
                strcat(&ErrorMessage[0], " ");
                strcat(&ErrorMessage[0], "in oen\n");
                yyerror(&ErrorMessage[0]);
                return 1;
               }
}
```

Выводы

В рамках четвертого этапа курсовой работы по разработке компилятора с языка PL1 (в рамках задания) были исследованы возможности лексического FLEX и синтаксического Bison анализаторов. На основе полученных данных был разработан компилятор, формирующий из исходного кода задания код на языке ассемблер. Результатом выполнения стало установление правильности разработанного компилятора.

Стоит отметить, что в данной конкретной задаче использование технологий flex/bison является довольно удобным, однако при решении более сложных задач могут возникнуть трудности, связанные с построением нужного анализатора и неудобной формой отладки.

Таким образом можно говорить об успешном выполнении курсовой работы по дисциплине «Системы программирования».