

Минобрнауки России  
Санкт-Петербургский государственный политехнический университет  
Институт информационных технологий и управления  
**Кафедра «Информационные и управляющие системы»**

## **КУРСОВОЙ ПРОЕКТ**

Разработка учебной системы программирования

В а р и а н т 7

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

Минобрнауки России  
Санкт-Петербургский государственный политехнический университет  
Институт информационных технологий и управления  
Кафедра «Информационные и управляющие системы»

## **КУРСОВОЙ ПРОЕКТ**

Разработка учебной системы программирования

В а р и а н т 7

**Построение компилятора с ЯВУ**

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 .....                         | 4  |
| План работы.....                      | 4  |
| Эквивалент на Ассемблере .....        | 5  |
| Модификация грамматики языка .....    | 6  |
| Матрица смежности.....                | 10 |
| Таблица синтаксических правил .....   | 11 |
| Таблица входов в правила .....        | 13 |
| Модификация функций компилятора ..... | 14 |
| Выводы.....                           | 20 |

## Задание

Усовершенствовать компилятор с языка высокого уровня (ЯВУ) 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;
```

## План работы

Необходимо доработать компилятор с ЯВУ, дополнив его новой функциональностью. В новую функциональность входят:

1. новый тип данных (bit);
2. операция конкатенации (!!);
3. операция взятия подстроки битовой строки (substr).

Для решения этой задачи необходимо:

1. модифицировать грамматику языка;
2. изменить и дополнить матрицу смежности;
3. расширить таблицу синтаксических правил;
4. расширить таблицу входов в правила;
5. добавить и изменить необходимые функции компилятора.

## 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.

## Модификация грамматики языка

Ниже представлена грамматика в модифицированном виде. Серым выделены добавленные правила:

1. `<PRO> ::= <OPR><TEL><OEN>`
2. `<OPR> ::= <IPR>:PROC_OPTIONS (MAIN);`
3. `<IPR> ::= <IDE>`
4. `<IDE> ::= <BUK> | <IDE><BUK> | <IDE><CIF>`
5. `<BUK> ::= A | B | C | D | E | M | P | X`
6. `<CIF> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`
7. `<TEL> ::= <ODC> | <TEL><ODC> | <TEL><OPA>`  
`| <TEL><OPL>`
8. `<ODC> ::= DCL_<IPE>_BIN_FIXED (<RZR>); |`  
`DCL_<IPE>_BIN_FIXED (<RZR>) INIT (<LIT>);`  
`| DCL_<IPE>_BIT (<RZR>); | DCL_<IPE>_BIT (<RZR>) INIT (<LIT>);`
9. `<IPE> ::= <IDE>`
10. `<RZR> ::= <CIF> | <RZR><CIF>`
11. `<LIT> ::= <MAN>B`
12. `<MAN> ::= 1 | <MAN>0 | <MAN>1`
13. `<OPA> ::= <IPE>=<AVI>;`
- 13.1 `<OPL> ::= <IPE>=<LVI>;`
14. `<AVI> ::= <LIT> | <IPE> | <AVI><ZNK><LIT> |`  
`<AVI><ZNK><IPE>`
- 14.1 `<LVI> ::= <IPE> | <LVI><ZKNL><IPE> | <SSTR> | (<LVI>)`
15. `<ZNK> ::= + | -`
- 15.1 `<ZKNL> ::= !!`
- 15.2 `<SSTR> ::= SUBSTR (<LVI>, <RZR>, <RZR>)`
16. `<OEN> ::= END_<IPR>`

Здесь использованы следующие метасимволы и символы:

- "<" и ">" - левый и правый ограничители нетерминального символа,
- "::=" - метасимвол со смыслом "равно по определению",
- "|" - метасимвол альтернативного определения "или",
- "\_" - терминальный символ со смыслом "пробел",
- "<PRO>" - нетерминал "программа",
- "<OPR>" - нетерминал "оператор пролога программы",
- "<IPR>" - нетерминал "имя программы",
- "<IDE>" - нетерминал "идентификатор",
- "<BUK>" - нетерминал "буква",
- "<CIF>" - нетерминал "цифра",
- "<TEL>" - нетерминал "тело программы",
- "<ODC>" - нетерминал "оператор declare",
- "<IPE>" - нетерминал "имя переменной",
- "<RZR>" - нетерминал "разрядность",
- "<LIT>" - нетерминал "литерал",
- "<MAN>" - нетерминал "мантисса",
- "<OPA>" - нетерминал "оператор присваивания арифметический",
- "<OPL>" - нетерминал "оператор присваивания логический",
- "<AVI>" - нетерминал "арифметическое выражение",
- "<LVI>" - нетерминал "логическое выражение",
- "<ZNK>" - нетерминал "знак",
- "<ZMKL>" - нетерминал "знак логический",
- "<LOP>" - нетерминал "логическая операция",
- "<OEN>" - нетерминал "оператор эпилога программы".
- "<SUS>" - нетерминал "оператор подстрока",

Прежде чем приступать к редактированию матрицы смежности, представим данные правила в виде продукции. Для этого «перевернём» их слева направо:

1. <OPR><TEL><OEN> -> <PRO>
2. <IPR>:PROC\_OPTIONS(MAIN); -> <OPR>
3. <IDE> -> <IPR>
4. <BUK> -> <IDE>
5. <IDE><BUK> -> <IDE>
6. <IDE><CIF> -> <IDE>
7. A -> <BUK>
8. B -> <BUK>
9. C -> <BUK>
10. D -> <BUK>
11. E -> <BUK>
12. M -> <BUK>
13. P -> <BUK>
14. X -> <BUK>
15. 0 -> <CIF>
16. 1 -> <CIF>
17. 2 -> <CIF>
18. 3 -> <CIF>
19. 4 -> <CIF>
20. 5 -> <CIF>
21. 6 -> <CIF>
22. 7 -> <CIF>
23. 8 -> <CIF>
24. 9 -> <CIF>
25. <ODC> -> <TEL>
26. <TEL><ODC> -> <TEL>
27. <TEL><OPA> -> <TEL>
28. <TEL><OPL> -> <TEL>
28. DCL\_IPE\_BIN\_FIXED(<RZR>); -> <ODC>
29. DCL\_IPE\_BIN\_FIXED(<RZR>) INIT(<LIT>); -> <ODC>
30. DCL\_IPE\_BIT(<RZR>); -> <ODC>
31. DCL\_IPE\_BIT(<RZR>) INIT(<LIT>); -> <ODC>
32. <IDE> -> <IPE>
33. <CIF> -> <RZR>
34. <RZR><CIF> -> <RZR>
35. <MAN>B -> <LIT>
36. 1 -> <MAN>
37. <MAN>0 -> <MAN>
38. <MAN>1 -> <MAN>
39. <IPE>=<AVI>; -> <OPA>
40. <IPE>=<LVI>; -> <OPL>
41. <LIT> -> <AVI>
42. <IPE> -> <AVI>
43. <AVI><ZNK><LIT> -> <AVI>
44. <AVI><ZNK><IPE> -> <AVI>
45. <IPE> -> <LVI>
46. <LVI><ZNKL><IPE> -> <LVI>
47. <SSTR> -> <LVI>
48. (<LVI>) -> <LVI>
49. + -> <ZNK>
50. - -> <ZNK>
51. !! -> <ZNKL>
52. SUBSTR(<LVI>, <RZR>, <RZR>) -> <SSTR>
53. END\_IPR -> <OEN>



Теперь, просматривая каждую из продукций слева-направо, сгруппируем продукции, имеющие общие части в "кусты", в которых роль "ствола" играют общие части продукций, а роль "ветвей" – различающиеся части продукций.

1. <OPR><TEL><OEN> -> <PRO>
2. <IPR>:PROC\_OPTIONS(MAIN); -> <OPR>
3. <IDE> -> <BUK> -> <IDE>
  - L-> <CIF> -> <IDE>
  - L-> <IPR>
  - L-> <IPE>
4. <BUK> -> <IDE>
5. A -> <BUK>
6. B -> <BUK>
7. C -> <BUK>
8. D -> CL\_<IPE>\_BIN\_FIXED(<RZR>)INIT(<LIT>); -> <ODC>
  - | L-> ; -> <ODC>
  - | L->T(<RZR>)INIT(<LIB>); -> <ODC>
  - | L-> ; -> <ODC>
  - L <BUK>
9. E -> ND\_<IPR> -> <OEN>
  - L-> <BUK>
10. M -> <BUK>
11. P -> <BUK>
12. X -> <BUK>
13. 0 -> <CIF>
14. 1 -> <CIF>
  - L-> <MAN>
15. 2 -> <CIF>
16. 3 -> <CIF>
17. 4 -> <CIF>
18. 5 -> <CIF>
19. 6 -> <CIF>
20. 7 -> <CIF>
21. 8 -> <CIF>
22. 9 -> <CIF>
23. <ODC> -> <TEL>
24. <TEL> -> <ODC> -> <TEL>
  - L-> <OPA> -> <TEL>
  - L-> <OPL> -> <TEL>
25. <CIF> -> <RZR>
26. <RZR><CIF> -> <RZR>
27. <MAN> -> B -> <LIT>
  - L-> 0 -> <MAN>
  - L-> 1 -> <MAN>
28. <IPE> -> =<AVI>; -> <OPA>
  - | L-> <LVI>; -> <OPL>
  - L-> <AVI>
  - L-> <LVI>
29. <LIT> -> <AVI>
30. <AVI><ZNK> -> <LIT> -> <AVI>
  - L-> <IPE> -> <AVI>
31. <LVI><ZKNL><IPE> -> <LVI>
32. <SSSTR> -> <LVI>
33. (<LVI>) -> <LVI>
34. + -> <ZNK>
35. - -> <ZNK>
36. !! -> <ZKNL>
37. SUBSTR(<LVI>, <RZR>, <RZR>) -> <SSSTR>

Таким образом, в грамматику языка были добавлены нетерминалы OPL (операнд присваивания логический), LVI (логическое выражение), ZKL (знак логический) и SUS (подстрока) и терминалы U, ! и запятая.

## Матрица смежности

Были исправлены некоторые глобальные переменные:

```
#define NVXOD      60    /* - табл.входов;          */
```

```
#define NNETRM     20    /* - списка нетерминалов; */
```

Ниже представим изменённые фрагменты матрицы смежности (заголовки добавленных строк и столбцов выделены серым):

```
| AVI:BUK:CIF:IDE:IFE:IPR:LIT:MAN:ODC:OEN:OPA:OPR:PRO:RZR:TEL:ZNK:OPL:LVI:ZKL:SUS |
| :   :   :   :   :   :   :   :   :   :   :   :   :   :   :   :   :   :   :   |
{ /*IFE*/ 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 },
{ /*OPL*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 },
{ /*LVI*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 },
{ /*ZKL*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 },
{ /*SUS*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 },
{ /* S*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 },
{ /* (**/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 },
{ /* U*/ 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 },
{ /* !*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 },
{ /* ,*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 }
...

```

Единица в матрице означает, что правило, начинающееся с терминала или нетерминала в строке, завершится нетерминалом в столбце. Например, правило, начинающееся с «S» завершится нетерминалом «SUS».

## Таблица синтаксических правил

Была исправлена глобальная переменная:

```
#define NSINT      262    /* - табл.синтакс.правил; */
```

Добавленные и изменённые фрагменты таблицы синтаксических правил приводятся ниже:

|             | NN  | :  | посл | : | пред | : | дер            | :     | альт  |    |
|-------------|-----|----|------|---|------|---|----------------|-------|-------|----|
| /*          |     |    |      |   |      |   | вход с символа | - D   |       | */ |
| {/*.        | 42  | ./ | 43   | , | 0    | , | "D "           | ,     | 0 }   |    |
| {/*.        | 43  | ./ | 44   | , | 42   | , | "BUK"          | ,     | 45 }  |    |
| {/*.        | 44  | ./ | 0    | , | 43   | , | "* "           | ,     | 0 }   |    |
| ...         |     |    |      |   |      |   |                |       |       |    |
| {/*.        | 45  | ./ | 46   | , | 42   | , | "C "           | ,     | 0 }   |    |
| {/*.        | 46  | ./ | 47   | , | 45   | , | "L "           | ,     | 0 }   |    |
| {/*.        | 47  | ./ | 48   | , | 46   | , | " "            | ,     | 0 }   |    |
| {/*.        | 48  | ./ | 49   | , | 47   | , | "IDE"          | ,     | 0 }   |    |
| {/*.        | 49  | ./ | 50   | , | 48   | , | " "            | ,     | 0 }   |    |
| {/*.        | 50  | ./ | 51   | , | 49   | , | "B "           | ,     | 187 } |    |
| {/*.        | 51  | ./ | 52   | , | 50   | , | "I "           | ,     | 0 }   |    |
| {/*.        | 52  | ./ | 53   | , | 51   | , | "N "           | ,     | 201 } |    |
| {/*.        | 53  | ./ | 54   | , | 52   | , | " "            | ,     | 0 }   |    |
| {/*.        | 54  | ./ | 55   | , | 53   | , | "F "           | ,     | 0 }   |    |
| {/*.        | 55  | ./ | 56   | , | 54   | , | "I "           | ,     | 0 }   |    |
| {/*.        | 56  | ./ | 57   | , | 55   | , | "X "           | ,     | 0 }   |    |
| {/*.        | 57  | ./ | 58   | , | 56   | , | "E "           | ,     | 0 }   |    |
| {/*.        | 58  | ./ | 59   | , | 57   | , | "D "           | ,     | 0 }   |    |
| ...         |     |    |      |   |      |   |                |       |       |    |
| /*          |     |    |      |   |      |   | вход с символа | - IPE |       | */ |
| {/*.        | 153 | ./ | 154  | , | 0    | , | "IPE"          | ,     | 0 }   |    |
| {/*.        | 154 | ./ | 155  | , | 153  | , | "= "           | ,     | 159 } |    |
| {/*.        | 155 | ./ | 156  | , | 154  | , | "AVI"          | ,     | 218 } |    |
| {/*.        | 156 | ./ | 157  | , | 155  | , | "; "           | ,     | 0 }   |    |
| {/*.        | 157 | ./ | 158  | , | 156  | , | "OPA"          | ,     | 0 }   |    |
| {/*.        | 158 | ./ | 0    | , | 157  | , | "* "           | ,     | 0 }   |    |
| {/*.        | 159 | ./ | 160  | , | 153  | , | "AVI"          | ,     | 222 } |    |
| {/*.        | 160 | ./ | 0    | , | 159  | , | "* "           | ,     | 0 }   |    |
| ...         |     |    |      |   |      |   |                |       |       |    |
| /*.         |     |    |      |   |      |   | вход с символа | - TEL |       | */ |
| {/*.        | 179 | ./ | 180  | , | 0    | , | "TEL"          | ,     | 0 }   |    |
| {/*.        | 180 | ./ | 181  | , | 179  | , | "ODC"          | ,     | 183 } |    |
| {/*.        | 181 | ./ | 182  | , | 180  | , | "TEL"          | ,     | 0 }   |    |
| {/*.        | 182 | ./ | 0    | , | 181  | , | "* "           | ,     | 0 }   |    |
| {/*.        | 183 | ./ | 184  | , | 179  | , | "OPA"          | ,     | 224 } |    |
| {/*.        | 184 | ./ | 185  | , | 183  | , | "TEL"          | ,     | 0 }   |    |
| {/*.        | 185 | ./ | 0    | , | 184  | , | "* "           | ,     | 0 }   |    |
| ...         |     |    |      |   |      |   |                |       |       |    |
| /* ***** */ |     |    |      |   |      |   |                |       |       |    |
| {/*.        | 201 | ./ | 202  | , | 51   | , | "T "           | ,     | 0 }   |    |

```

{/*. 202 .*/ 203 , 201 , "( " , 0 },
{/*. 203 .*/ 204 , 202 , "RZR" , 0 },
{/*. 204 .*/ 205 , 203 , ") " , 0 },
{/*. 205 .*/ 206 , 204 , ";" , 208 },
{/*. 206 .*/ 207 , 205 , "ODC" , 0 },
{/*. 207 .*/ 208 , 206 , "*" , 0 },

{/*. 208 .*/ 209 , 204 , "I " , 0 },
{/*. 209 .*/ 210 , 208 , "N " , 0 },
{/*. 210 .*/ 211 , 209 , "I " , 0 },
{/*. 211 .*/ 212 , 210 , "T " , 0 },
{/*. 212 .*/ 213 , 211 , "( " , 0 },
{/*. 213 .*/ 214 , 212 , "LIT" , 0 },
{/*. 214 .*/ 215 , 213 , ") " , 0 },
{/*. 215 .*/ 216 , 214 , ";" , 0 },
{/*. 216 .*/ 217 , 215 , "ODC" , 0 },
{/*. 217 .*/ 0 , 216 , "*" , 0 },

{/*. 218 .*/ 219 , 154 , "LVI" , 0 },
{/*. 219 .*/ 220 , 218 , ";" , 0 },
{/*. 220 .*/ 221 , 219 , "OPL" , 0 },
{/*. 221 .*/ 0 , 220 , "*" , 0 },

{/*. 222 .*/ 223 , 153 , "LVI" , 0 },
{/*. 223 .*/ 0 , 222 , "*" , 0 },

{/*. 224 .*/ 225 , 179 , "OPL" , 0 },
{/*. 225 .*/ 226 , 224 , "TEL" , 0 },
{/*. 226 .*/ 0 , 225 , "*" , 0 },

/*
                               ВХОД С СИМВОЛА - LVI */
{/*. 227 .*/ 228 , 0 , "LVI" , 0 },
{/*. 228 .*/ 229 , 227 , "ZKL" , 0 },
{/*. 229 .*/ 230 , 228 , "IPE" , 0 },
{/*. 230 .*/ 231 , 229 , "LVI" , 0 },
{/*. 231 .*/ 0 , 230 , "*" , 0 },

/*
                               ВХОД С СИМВОЛА - SUS */
{/*. 232 .*/ 233 , 0 , "SUS" , 0 },
{/*. 233 .*/ 234 , 232 , "LVI" , 0 },
{/*. 234 .*/ 0 , 233 , "*" , 0 },

/*
                               ВХОД С СИМВОЛА - ( */
{/*. 235 .*/ 236 , 0 , "( " , 0 },
{/*. 236 .*/ 237 , 235 , "LVI" , 0 },
{/*. 237 .*/ 238 , 236 , ") " , 0 },
{/*. 238 .*/ 239 , 237 , "LVI" , 0 },
{/*. 239 .*/ 0 , 238 , "*" , 0 },

/*
                               ВХОД С СИМВОЛА - S */
{/*. 240 .*/ 241 , 0 , "S " , 0 },
{/*. 241 .*/ 242 , 240 , "U " , 0 },
{/*. 242 .*/ 243 , 241 , "B " , 0 },
{/*. 243 .*/ 244 , 242 , "S " , 0 },
{/*. 244 .*/ 245 , 243 , "T " , 0 },
{/*. 245 .*/ 246 , 244 , "R " , 0 },
{/*. 246 .*/ 247 , 245 , "( " , 0 },
{/*. 247 .*/ 248 , 246 , "LVI" , 0 },
{/*. 248 .*/ 249 , 247 , " , " , 0 },
{/*. 249 .*/ 250 , 248 , "RZR" , 0 },
{/*. 250 .*/ 251 , 249 , " , " , 0 },
{/*. 251 .*/ 252 , 250 , "RZR" , 0 },
{/*. 252 .*/ 253 , 251 , ") " , 0 },
{/*. 253 .*/ 254 , 252 , "SUS" , 0 },

```

```

{ /*. 254      .*/      0 , 253 , "*" , 0 },

/*
                               ВХОД С СИМВОЛА - !      */
{ /*. 255      .*/      256 , 0 , "!" , 0 },
{ /*. 256      .*/      257 , 255 , "!" , 0 },
{ /*. 257      .*/      258 , 256 , "ZKL" , 0 },
{ /*. 258      .*/      0 , 257 , "*" , 0 },

/*
                               ВХОД С СИМВОЛА - U      */
{ /*. 259      .*/      260 , 0 , "U" , 0 },
{ /*. 260      .*/      261 , 259 , "BUK" , 0 },
{ /*. 261      .*/      0 , 260 , "*" , 0 }

```

Добавленные строки таблицы описывают новые правила языка.

### Таблица входов в правила

Ниже представлен фрагменты таблицы входов в правила, введенные в соответствии с новыми возможностями языка:

|   | NN | СИМВОЛ | ВХОД | ТИП |
|---|----|--------|------|-----|
| { /*. 21      .*/      "OPL" , 0 , 'N' },   |    |        |      |     |
| { /*. 22      .*/      "LVI" , 227 , 'N' }, |    |        |      |     |
| { /*. 23      .*/      "ZKL" , 0 , 'N' },   |    |        |      |     |
| { /*. 24      .*/      "SUS" , 232 , 'N' }, |    |        |      |     |
| ...   |    |        |      |     |
| { /*. 52      .*/      "S " , 240 , 'T' },  |    |        |      |     |
| { /*. 53      .*/      "( " , 235 , 'T' },  |    |        |      |     |
| { /*. 62      .*/      "U " , 259 , 'T' },  |    |        |      |     |
| { /*. 63      .*/      "!" , 255 , 'T' },   |    |        |      |     |
| { /*. 64      .*/      ", " , 0 , 'T' }     |    |        |      |     |

## Модификация функций компилятора

С целью расширения функциональности языка в следующие функции компилятора были внесены изменения:

### 1) Функция уплотнения

```
void compress_ISXTXT()
{
    I3 = 0;
    char current = 0;
    for (I1 = 0; I1 < NISXTXT; I1++) {
        for (I2 = 0; I2 < 80; I2++){
            current = ISXTXT[I1][I2];
            if (current == '\x0') {
                break;
            }
            if (current == '\n') {
                continue;
            }
            if (current == ' ' && (PREDSYM == ' ' || PREDSYM == ';' ||
                || PREDSYM == ')') || PREDSYM == ':' || PREDSYM == '(')) {
                PREDSYM = current;
                continue;
            }
            if ((current == '!' || current == '+' || current == '-' ||
                || current == '=' || current == '(' ||
                || current == ')' || current == '*')
                && PREDSYM == ' ') {
                I3--;
                PREDSYM = current;
                STROKA[I3] = PREDSYM;
                I3++;
                continue;
            }
            if (current == ' ' && (PREDSYM == '+' || PREDSYM == '-' ||
                || PREDSYM == '=' || PREDSYM == '*' || PREDSYM == '!')) {
                continue;
            }
            PREDSYM=current;
            STROKA[I3]=current;
            I3++;
        }
    }
    STROKA[I3] = '\x0';
}
```

### 2) Функция формирования лексем из уплотненного текста

```
void FORM()
{
    int i, j;
    for (IFORMT = 0; IFORMT < MAXFORMT; IFORMT++)
        memcpy(FORMT[IFORMT], "\x0\x0\x0\x0\x0\x0\x0\x0\x0", 9);
    IFORMT = 0;
    j = DST[I2].DST2;
    FORM1:
    for (i = j; i <= DST[I2].DST4 + 1; i++) {
        if (STROKA[i] == ':' || STROKA[i] == ' ' || STROKA[i] == '(' || STROKA[i] == '!' ||
            || STROKA[i] == ')') || STROKA[i] == ';' || STROKA[i] == '+' ||
            || STROKA[i] == '-' || STROKA[i] == '=' || STROKA[i] == '*') {
            FORMT[IFORMT][i - j] = '\x0';
            IFORMT++;
            j = i + 1;
            goto FORM1;
        } else
            FORMT[IFORMT][i - j] = STROKA[i];
    }
    return;
}
```

### 3) Функция вычисления нетерминала ODC (оп. DCL) на первом проходе

```

int ODC1() {
    int i;
    FORM();
    for (i = 0; i < ISYM; i++)
    {
        if (!strcmp(SYM[i].NAME, FORMT[1]) && strlen(SYM[i].NAME) == strlen(FORMT[1]))
            return 6;
    }
    strcpy(SYM[ISYM].NAME, FORMT[1]);
    strcpy(SYM[ISYM].RAZR, FORMT[4]);
    /* фикатора запоминаем его*/
    /* вместе с разрядностью в*/
    /* табл.SYM */

    if (!strcmp(FORMT[2], "BIN") && !strcmp(FORMT[3], "FIXED")
    {
        SYM[ISYM].TYPE = 'B';
        goto ODC11;
    } else if (!strcmp(FORMT[2], "BIT"))
    {
        SYM[ISYM].TYPE = 'L';
        goto ODC12;
    } else
    {
        SYM[ISYM].TYPE = 'U';
        return 2;
    }
    ODC11:
    if (!strcmp(FORMT[5], "INIT"))
        strcpy(SYM[ISYM++].INIT, FORMT[6]);
    else
        strcpy(SYM[ISYM++].INIT, "0B");
    return 0;
    ODC12:
    strcpy(SYM[ISYM].RAZR, FORMT[3]);
    if (!strcmp(FORMT[4], "INIT"))
        strcpy(SYM[ISYM++].INIT, FORMT[5]);
    else
        strcpy(SYM[ISYM++].INIT, "0B");
    return 0;
}

```

#### 4) Функция формирования пролога

```

int OPR2() {
    char i = 0;
    FORM();
    while (FORMT[0][i] != '\x0')
        ASS_CARD._BUFCARD.METKA[i++] = FORMT[0][i];
    memcpy(ASS_CARD._BUFCARD.OPERAC, "START", 5);
    memcpy(ASS_CARD._BUFCARD.OPERAND, "0", 1);
    memcpy(ASS_CARD._BUFCARD.COMM, "Nacalo programmy", 16);
    ZKARD();
    memcpy(ASS_CARD._BUFCARD.OPERAC, "BALR", 4);
    memcpy(ASS_CARD._BUFCARD.OPERAND, "RBASE,0", 7);
    memcpy(ASS_CARD._BUFCARD.COMM, "Zagruzit' registr bazy", 22);
    ZKARD();
    memcpy(ASS_CARD._BUFCARD.OPERAC, "USING", 5);
    memcpy(ASS_CARD._BUFCARD.OPERAND, "*,RBASE", 7);
    memcpy(ASS_CARD._BUFCARD.COMM, "Naznagit' registr bazoy", 23);
    ZKARD();
    return 0;
}

```

## 5) Функция вычисления нетерминала OEN (оп. END), формирование эпилога

```

int OEN2() {
    char RAB[20];
    char i = 0;
    FORM();
    memcpy(ASS_CARD._BUFCARD.OPERAC, "BCR", 3);
    memcpy(ASS_CARD._BUFCARD.OPERAND, "15,RVIX", 7);
    memcpy(ASS_CARD._BUFCARD.COMM,
"Vyhod iz programmy", 18);
    ZKARD();
    for (i = 0; i < ISYM; i++) {
        if (isalpha ( SYM [i].NAME [0] ))
        {
            if (SYM[i].TYPE == 'B')
            {
                strcpy(ASS_CARD._BUFCARD.METKA, SYM[i].NAME);
                ASS_CARD._BUFCARD.METKA[strlen(ASS_CARD._BUFCARD.METKA)] = ' ';
                memcpy(ASS_CARD._BUFCARD.OPERAC, "DC", 2);
                if (strcmp(SYM[i].RAZR, "15") <= 0)
                    strcpy(ASS_CARD._BUFCARD.OPERAND, "H\\");
                else
                    strcpy(ASS_CARD._BUFCARD.OPERAND, "F\\");
                strcat(ASS_CARD._BUFCARD.OPERAND, gcvt(VALUE(SYM[i].INIT), 10,
&RAB[0]));
                ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] =
'\\';
                memcpy(ASS_CARD._BUFCARD.COMM,
"Определение переменной", 22);
                ZKARD();
            } else if (SYM[i].TYPE == 'L')
            {
                strcpy(ASS_CARD._BUFCARD.METKA, SYM[i].NAME);
                ASS_CARD._BUFCARD.METKA[strlen(ASS_CARD._BUFCARD.METKA)] = ' ';
                int a = 0;
                while (SYM[i].INIT[a] != 'B') {
                    a++;
                }
                SYM[i].INIT[a] = 0;

                if (a == 1) {
                    memcpy(ASS_CARD._BUFCARD.OPERAC, "DS", 2);
                    strcpy(ASS_CARD._BUFCARD.OPERAND, "0H");
                    ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)]
=
                    ' ';
                    ZKARD();
                    memcpy(ASS_CARD._BUFCARD.OPERAC, "DS", 2);
                    memcpy(ASS_CARD._BUFCARD.OPERAND, "BL2", 3);
                    ZKARD();
                    strcpy(ASS_CARD._BUFCARD.METKA, "TMP");
                    ASS_CARD._BUFCARD.METKA[strlen(ASS_CARD._BUFCARD.METKA)] = '
';

                    memcpy(ASS_CARD._BUFCARD.OPERAC, "DS", 2);
                    strcpy(ASS_CARD._BUFCARD.OPERAND, "0H");
                    ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)]
=
                    ' ';

                    ZKARD();
                    memcpy(ASS_CARD._BUFCARD.OPERAC, "DC", 2);
                    strcpy(ASS_CARD._BUFCARD.OPERAND, "BL2\\");
                    strcat(ASS_CARD._BUFCARD.OPERAND, "111");
                    strcat(ASS_CARD._BUFCARD.OPERAND, "\\");
                    ASS_CARD._BUFCARD.OPERAND[strlen
(ASS_CARD._BUFCARD.OPERAND)] = ' ';
                    ZKARD();
                } else {
                    memcpy(ASS_CARD._BUFCARD.OPERAC, "DS", 2);
                    strcpy(ASS_CARD._BUFCARD.OPERAND, "0H");
                    ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)]
=
                    ' ';

                    ZKARD();
                    memcpy(ASS_CARD._BUFCARD.OPERAC, "DC", 2);
                    strcpy(ASS_CARD._BUFCARD.OPERAND, "BL2\\");
                    strcat(ASS_CARD._BUFCARD.OPERAND, SYM[i].INIT);
                    ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)]
=
                    '\\';

                    ZKARD();
                }
            }
        }
    }
}

```



```

    }
    memcpy(ASS_CARD._BUFCARD.METKA, "RBASE", 5);
    memcpy(ASS_CARD._BUFCARD.OPERAC, "EQU", 3);
    memcpy(ASS_CARD._BUFCARD.OPERAND, "5", 1);
    ZKARD();
    memcpy(ASS_CARD._BUFCARD.METKA, "RVIX", 4);
    memcpy(ASS_CARD._BUFCARD.OPERAC, "EQU", 3);
    memcpy(ASS_CARD._BUFCARD.OPERAND, "14", 2);
    ZKARD();
    memcpy(ASS_CARD._BUFCARD.OPERAC, "END", 3); /
    i = 0;
    while (FORMT[1][i] != '\x0') ASS_CARD._BUFCARD.OPERAND[i] = FORMT[1][i++];
    memcpy(ASS_CARD._BUFCARD.COMM, "Konec programy", 15);
    ZKARD();
    return 0;
}

```

## 6) Функция формирования лексем из уплотненного текста

```

int gen_COD()
{
    int NOSH;
    int (* FUN [NNETRM][2]) () =
    {
        { /* 1 */ AVI1, AVI2 },
        { /* 2 */ BUK1, BUK2 },
        { /* 3 */ CIF1, CIF2 },
        { /* 4 */ IDE1, IDE2 },
        { /* 5 */ IPE1, IPE2 },
        { /* 6 */ IPR1, IPR2 },
        { /* 7 */ LIT1, LIT2 },
        { /* 8 */ MAN1, MAN2 },
        { /* 9 */ ODC1, ODC2 },
        { /* 10 */ OEN1, OEN2 },
        { /* 11 */ OPA1, OPA2 },
        { /* 12 */ OPR1, OPR2 },
        { /* 13 */ PRO1, PRO2 },
        { /* 14 */ RZR1, RZR2 },
        { /* 15 */ TEL1, TEL2 },
        { /* 16 */ ZNK1, ZNK2 },
        { /* 17 */ OPL1, OPL2 },
        { /* 18 */ LVI1, LVI2 },
        { /* 19 */ ZKL1, ZKL2 },
        { /* 20 */ SUS1, SUS2 }
    };
    for (I2 = 0; I2 < L; I2++)
        if ((NOSH = FUN[numb(DST[I2].DST1, 3)][0]()) != 0)
            return (NOSH);
    for ( I2 = 0; I2 < L; I2++ )
        if ( ( NOSH = FUN [numb ( DST [I2].DST1, 3 )][1]()) != 0)
            return (NOSH);
    return 0;
}

```

Кроме того были добавлены следующие новые функции:

### 1) Оператор логического присваивания на первом проходе

```

int OPL1() {
    return 0;
}

```

### 2) Оператор логического присваивания на втором проходе

```

int OPL2() {
    int i;
    FORM();
    for (i = 0; i < ISYM; i++) {
        if (!strcmp(SYM[i].NAME, FORMT[0]) && strlen(SYM[i].NAME) == strlen(FORMT[0])) {
            if (SYM[i].TYPE == 'L')
            {
                memcpy(ASS_CARD._BUFCARD.OPERAC, "STH", 3);
                strcpy(ASS_CARD._BUFCARD.OPERAND, "3,");
            }
        }
    }
}

```

```

        strcat(ASS_CARD._BUFCARD.OPERAND, FORMT[0]);
        ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = ' ';
        memcpy(ASS_CARD._BUFCARD.COMM, "Formirovaniye znaceniya
logic.virazeniya", 37);

        ZKARD();
        return 0;
    }
    else
        return 3;
}
}
return 4;
}

```

### 3) Функция логического выражения на первом проходе

```

int LVI1() {
    return 0;
}

```

### 4) Функция логического выражения на втором проходе

```

int LVI2() {
    char i;
    FORM();
    if (IFORMT == 1)
    {
        for (i = 0; i < ISYM; i++)
        {
            if (!strcmp(SYM[i].NAME, FORMT[0]) && strlen(SYM[i].NAME) ==
strlen(FORMT[0])) {
                if (SYM[i].TYPE == 'L')
                {
                    memcpy(ASS_CARD._BUFCARD.OPERAC, "LH", 2);
                    strcpy(ASS_CARD._BUFCARD.OPERAND, "3,");
                    strcat(ASS_CARD._BUFCARD.OPERAND, FORMT[0]);
                    ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)]
= ' ';
                    memcpy(ASS_CARD._BUFCARD.COMM, "Zagruzka peremennoy v
registr", 29);

                    ZKARD();
                    return 0;
                }
                else
                    return 3;
            }
        }
        return 4; /
    }
    else
    {
        for (i = 0; i < ISYM; i++)
        {
            if(STROKA[DST[I2].DST4] == ' ')
                return 0;
            if (!strcmp(SYM[i].NAME,
FORMT[IFORMT - 1]) && strlen(SYM[i].NAME) == strlen(FORMT[IFORMT-
1])) {
                memcpy(ASS_CARD._BUFCARD.OPERAC, "LH", 2);
                strcpy(ASS_CARD._BUFCARD.OPERAND, "4,");
                strcat(ASS_CARD._BUFCARD.OPERAND, FORMT[IFORMT - 1]);
                ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)]= ' ';
                memcpy(ASS_CARD._BUFCARD.COMM, "Formirovaniye promezutocnogo
znaceniya", 36);

                ZKARD();
                memcpy(ASS_CARD._BUFCARD.OPERAC, "SRL", 3);
                strcpy(ASS_CARD._BUFCARD.OPERAND, "4,");
                strcat(ASS_CARD._BUFCARD.OPERAND, "3");
                ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)]= ' ';
                memcpy(ASS_CARD._BUFCARD.COMM, "Sdvig operanda vpravo", 21);
                ZKARD();
                memcpy(ASS_CARD._BUFCARD.OPERAC, "OR", 2);
                strcpy(ASS_CARD._BUFCARD.OPERAND, "3,");
                strcat(ASS_CARD._BUFCARD.OPERAND, "4");
                ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)]= ' ';
            }
        }
    }
}

```

```

        memcpy(ASS_CARD._BUFCARD.COMM, "Logicheskoye ILI registrov", 26);
        ZKARD();
        return 0;
    } else
        return 3;
    }
}
return 4;
}
}

```

## 5) Функция логического знака на первом проходе

```

int ZKL1() {
    return 0;
}

```

## 6) Функция логического знака на втором проходе

```

int ZKL2() {
    return 0;
}

```

## 7) Функция вычисления SUS на первом проходе (оп. substr)

```

int SUS1() {
    return 0;
}

```

## 8) Функция вычисления SUS на втором проходе (оп. substr)

```

int SUS2() {
    memcpy(ASS_CARD._BUFCARD.OPERAC, "SLL", 3);
    strcpy(ASS_CARD._BUFCARD.OPERAND, "3,");
    strcat(ASS_CARD._BUFCARD.OPERAND, "1");
    ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = ' ';
    memcpy(ASS_CARD._BUFCARD.COMM, "Sdviig operanda vlevo", 20);
    ZKARD();
    memcpy(ASS_CARD._BUFCARD.OPERAC, "LH", 2);
    strcpy(ASS_CARD._BUFCARD.OPERAND, "4,");
    strcat(ASS_CARD._BUFCARD.OPERAND, "TMP");
    ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = ' ';
    memcpy(ASS_CARD._BUFCARD.COMM, "Zagruzka maski v reg", 20);
    ZKARD();
    memcpy(ASS_CARD._BUFCARD.OPERAC, "NR", 2);
    strcpy(ASS_CARD._BUFCARD.OPERAND, "3,");
    strcat(ASS_CARD._BUFCARD.OPERAND, "4");
    ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = ' ';
    memcpy(ASS_CARD._BUFCARD.COMM, "Logicheskoye I registr", 21);
    ZKARD();
    return 0;
}

```

## **Выводы**

В рамках первого этапа курсовой работы по написанию компилятора с языка высокого уровня в рамках поставленной задачи были выполнены все поставленные задачи:

- 1) Разработан код на ассемблере, выполняющий действия, эквивалентные описанным на языке PL1
- 2) Модифицирована грамматика и синтаксические правила языка
- 3) Существующий компилятор доработан с учетом новых правил

Таким образом можно говорить об успешном завершении первого этапа, результатом которого стал компилятор с языка PL1 на ассемблер в рамках задания.

В качестве самой трудоемкой задачи можно отметить разбор и изменение существующего кода компилятора. В качестве основных проблем можно отметить:

- 1) Избыточное количество глобальных переменных
- 2) Неудобные комментарии
  - ✓ Комментарии написаны на русском языке
  - ✓ Комментарии написаны по столбцам а не по строкам. При форматировании кода они съезжают
- 3) Большое количество операторов goto
- 4) Объявление методов и переменных по правилам транслитерации (было бы удобнее, если бы имена соответствовали английским названиям)

Минобрнауки России  
Санкт-Петербургский государственный политехнический университет  
Институт информационных технологий и управления  
**Кафедра «Информационные и управляющие системы»**

## **КУРСОВОЙ ПРОЕКТ**

Разработка учебной системы программирования

В а р и а н т 7

**Построение компилятора с Ассемблера**  
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 .....                      | 23 |
| План работы.....                   | 24 |
| Расширение таблицы команд.....     | 24 |
| Модификация кода компилятора ..... | 25 |
| Первый просмотр: .....             | 25 |
| Второй просмотр: .....             | 27 |
| Выводы.....                        | 30 |

## Задание

Воспользовавшись результатами первого этапа курсовой работы, доработать существующий компилятор с ассемблера для получения объектного представления программы. В дальнейшем результат будет использован в третьей части курсовой работы.

*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;
```

В результате выполнения первой части курсовой работы был получен эквивалент программы на языке Ассемблер.

|       |       |          |                                 |
|-------|-------|----------|---------------------------------|
| 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                 |

## План работы

Необходимо доработать компилятор с Ассемблера в объектное представление, дополнив его новой функциональностью. В новую функциональность входят:

4. поддержка новых команд (LH, STH, SRL, SLL, OR, NR);
5. поддержка типа BL2.

Для решения этой задачи необходимо:

6. расширить таблицу машинных команд;
7. модифицировать обработчики команд DC и DS первого и второго просмотров, а также изменить обработчик команд RX-типа.

## Расширение таблицы команд

```
#define DL_ASSTEXT 29
#define DL_OBJTEXT 100      /*длина об'ектн. текста */
#define NSYM 50             /*размер табл.символов */
#define NPOP 6              /*размер табл.псевдоопер. */
#define NOP 12              /*размер табл.операций */
```

Таблица машинных операций имеет следующий вид (добавленные фрагменты выделены цветом):

```
T_MOP [NOP] =
{
  {{ 'B', 'A', 'L', 'R', ' ' }} , '\x05' , 2 , FRR} ,
  {{ 'B', 'C', 'R', ' ', ' ' }} , '\x07' , 2 , FRR} ,
  {{ 'S', 'T', ' ', ' ', ' ' }} , '\x50' , 4 , FRX} ,
  {{ 'L', ' ', ' ', ' ', ' ', ' ' }} , '\x58' , 4 , FRX} ,
  {{ 'A', ' ', ' ', ' ', ' ', ' ' }} , '\x5A' , 4 , FRX} ,
  {{ 'S', ' ', ' ', ' ', ' ', ' ' }} , '\x5B' , 4 , FRX} ,
  {{ 'L', 'H', ' ', ' ', ' ', ' ' }} , '\x48' , 4 , FRX} ,
  {{ 'S', 'R', 'L', ' ', ' ', ' ' }} , '\x01' , 4 , FRX} ,
  {{ 'S', 'L', 'L', ' ', ' ', ' ' }} , '\x02' , 4 , FRX} ,
  {{ 'O', 'R', ' ', ' ', ' ', ' ' }} , '\x16' , 4 , FRR} ,
  {{ 'N', 'R', ' ', ' ', ' ', ' ' }} , '\x14' , 4 , FRR} ,
  {{ 'S', 'T', 'H', ' ', ' ', ' ' }} , '\x40' , 4 , FRX} ,
};
```

Также в начале второго просмотра были установлены указатели на программные обработчики новых команд:

CONT3:

```
T_MOP[0].BXPORG = SRR;
```



```

T_MOP[1].BXPLOG = SRR;
T_MOP[2].BXPLOG = SRX;
T_MOP[3].BXPLOG = SRX;
T_MOP[4].BXPLOG = SRX;
T_MOP[5].BXPLOG = SRX;
T_MOP[6].BXPLOG = SRX;
T_MOP[7].BXPLOG = SRX;
T_MOP[8].BXPLOG = SRX;
T_MOP[9].BXPLOG = SRR;
T_MOP[10].BXPLOG = SRR;
T_MOP[11].BXPLOG = SRX;

```

## Модификация кода компилятора

В данном разделе рассмотрены модифицированные функции компилятора с ассемблера.

### Первый просмотр:

#### 1) Определение оператора DC при первом просмотре

```

int FDC()
{
    if (PRNMET == 'Y')
    {
        if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'F')
        {
            T_SYM[ITSYM].DLSYM = 4;
            T_SYM[ITSYM].PRPER = 'R';
            if (CHADR % 4)
            {
                CHADR = (CHADR / 4 + 1) * 4;
            }
            T_SYM[ITSYM].ZNSYM = CHADR;
            PRNMET = 'N';
        } else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'H') {
            T_SYM[ITSYM].DLSYM = 2;
            T_SYM[ITSYM].PRPER = 'R';
            if (CHADR % 2)
            {
                CHADR = (CHADR / 2 + 1) * 2;
            }
            T_SYM[ITSYM].ZNSYM = CHADR;
            PRNMET = 'N';
        } else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'B'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1] == 'L'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2] == '2')
        {
            T_SYM[ITSYM].DLSYM = 2;
            T_SYM[ITSYM].PRPER = 'R';
            if (CHADR % 2)
            {
                CHADR = (CHADR / 2 + 1) * 2;
            }
            T_SYM[ITSYM].ZNSYM = CHADR;
        } else {
            CHADR += 2;
        }
        PRNMET = 'N';
    } else
        return (1);
}

```

```

        return 0;
    } else {
        if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'B'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1] == 'L'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2] == '2')
        {
            T_SYM[ITSYM].DLSYM = 2;
            T_SYM[ITSYM].PRPER = 'R';
            if (CHADR % 2) /* и, если CHADR не указ.*/
            {
                CHADR = (CHADR / 2 + 1) * 2;
                T_SYM[ITSYM].ZNSYM = CHADR;
            } else {
                CHADR += 2;
            }
            PRNMET = 'N';
        }
        return (0);
    }
}

```

## 2) Определение оператора DS при первом просмотре

```

int FDS
{
    if (PRNMET == 'Y')
    {
        if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'F')
        {
            T_SYM[ITSYM].DLSYM = 4;
            T_SYM[ITSYM].PRPER = 'R';
            if (CHADR % 4)
            {
                CHADR = (CHADR / 4 + 1) * 4;
                T_SYM[ITSYM].ZNSYM = CHADR;
            }
            CHADR += 4;
            PRNMET = 'N';
        } else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'B'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1] == 'L'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2] == '2')
        {
            T_SYM[ITSYM].DLSYM = 2;
            T_SYM[ITSYM].PRPER = 'R';
            if (CHADR % 2)
            {
                CHADR = (CHADR / 2 + 1) * 2;
            } else {
                CHADR += 2;
            }
            PRNMET = 'N';
        } else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == '0'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1] == 'H')
        {
            T_SYM[ITSYM].DLSYM = 2;
            T_SYM[ITSYM].PRPER = 'R';
            if (CHADR % 2)
            {
                CHADR = (CHADR / 2 + 1) * 2;
                T_SYM[ITSYM].ZNSYM = CHADR;
            }
            PRNMET = 'N';
        } else
            return (1);
        return 0;
    } else {
        if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'B'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1] == 'L'
            && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2] == '2')
        {
            T_SYM[ITSYM].DLSYM = 2;

```

```

        T_SYM[ITSYM].PRPER = 'R';
        if (CHADR % 2)
        {
            CHADR = (CHADR / 2 + 1) * 2;
        } else {
            CHADR += 2;
        }
        PRNMET = 'N';
    } else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == '0'
        && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1] == 'H')
    {
        T_SYM[ITSYM].DLSYM = 2;
        T_SYM[ITSYM].PRPER = 'R';
        if (CHADR % 2)
        {
            CHADR = (CHADR / 2 + 1) * 2;
            T_SYM[ITSYM].ZNSYM = CHADR;
        }
        PRNMET = 'N';
    } else
        return (1);
    }
    return (0);
}

```

### 3) Определение операции RX при первом просмотре

```

int FRX()
{
    CHADR = CHADR + 4;
    if (PRNMET == 'Y')
    {
        T_SYM[ITSYM].DLSYM = 4;
        T_SYM[ITSYM].PRPER = 'R';
    }
    return (0);
}

```

## Второй просмотр:

### 1) Определение оператора DC при втором просмотре

```

int SDC()
{
    char *RAB;
    RX.OP_RX.OP = 0;
    RX.OP_RX.R1X2 = 0;
    if (!memcmp(TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "F'", 2))
    {
        RAB = strtok ((char*) TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND + 2, "'");
        RX.OP_RX.B2D2 = atoi(RAB);
        RAB = (char *) &RX.OP_RX.B2D2;
        swab(RAB, RAB, 2);
        STXT(4);
    } else if (!memcmp(TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "H'", 2))
    {
        RAB = ( (char*) TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND + 2,
        "'");
        RR.OP_RR.OP = 0;
        RR.OP_RR.R1R2 = atoi(RAB);
        RAB = (char *) &RR.OP_RR.R1R2;
        STXT(2);
    }
    else if (!memcmp(TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "BL2'", 4))
    {
        RAB = strtok ((char*) TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND + 4, "'");
        int value = atoi(RAB);
        int len = strlen(RAB);
    }
}

```

```

        value <= (16 - len);
        RAB = (char *) &value;
        swab(RAB, RAB, 2);
        char buf[2] = { TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2], '\x0' };
        int bytes = atoi(buf);
        RR.OP_RR.OP = 0;
        RR.OP_RR.R1R2 = 0;
        memcpy(RR.BUF_OP_RR, &value, 2);
        STXT(2);
    } else
        return (1);
    return (0);
}

```

## 2) Определение оператора DS на втором просмотре

```

int SDS()
{
    if ( TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'F')
    {
        RX.OP_RX.OP = 0;
        RX.OP_RX.R1X2 = 0;
        RX.OP_RX.B2D2 = 0;
        STXT(4);
    } else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == 'B'
        && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1] == 'L'
        && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[2] == '2')
    {
        RR.OP_RR.OP = 0;
        RR.OP_RR.R1R2 = 0;
        STXT(2);
    } else if (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[0] == '0'
        && TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND[1] == 'H' )
    {
        if (CHADR % 2)
        {
            CHADR = (CHADR / 2 + 1) * 2;
        }
    } else
        return (1);
    return (0);
}

```

## 3) Определение операции RX на втором посмотре

```

int SRX()
{
    char *METKA;
    char *METKA1;
    char *METKA2;
    char *PTR;
    int DELTA;
    int ZNSYM;
    int NBASRG;
    int J; int I;
    unsigned char R1X2;
    int B2D2;
    RX.OP_RX.OP = T_MOP[I3].CODOP;
    METKA1 = strtok ( (char*) TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, ", " );
    METKA2 = strtok(NULL, " " );
    if (isalpha ( (int) *METKA1 ) || (int) *METKA1 == '@')
    {
        for (J = 0; J <= ITSYM; J++)
        {
            METKA = strtok((char*) T_SYM[J].IMSYM, " ");
            if (!strcmp(METKA, METKA1))
            {
                R1X2 = T_SYM[J].ZNSYM << 4;
                goto SRX1;
            }
        }
    }
}

```

```

        return (2);
    } else
    {
        R1X2 = atoi(METKA1) << 4;
    }

SRX1:

if (isalpha ( (int) *METKA2 ) || (int) *METKA2 == '@')
{
    for (J = 0; J <= ITSYM; J++)
    {
        METKA = strtok((char*) T_SYM[J].IMSYM, " ");
        if (!strcmp(METKA, METKA2))
        {
            NBASRG = 0;
            DELTA = 0xffff - 1;
            ZNSYM = T_SYM[J].ZNSYM;
            for (I = 0; I < 15; I++)
            {
                if (
                    T_BASR[I].PRDOST == 'Y'          &&          ZNSYM - T_BASR[I].SMESH
                    >= 0 && ZNSYM - T_BASR[I].SMESH < DELTA )
                {
                    NBASRG = I + 1;
                    DELTA = ZNSYM - T_BASR[I].SMESH;
                }
            }
            if (NBASRG == 0 || DELTA > 0xffff)
                return (5);
            else
            {
                B2D2 = NBASRG << 12;
                B2D2 = B2D2 + DELTA;
                PTR = (char *) &B2D2;
                swab(PTR, PTR, 2);
                RX.OP_RX.B2D2 = B2D2;
            }
            goto SRX2;
        }
    }
    return (2);
} else if (isdigit(METKA2[0])) {
    NBASRG = 0;
    DELTA = atoi(&METKA2[0]);
    B2D2 = NBASRG << 12;
    B2D2 = B2D2 + DELTA;
    PTR = (char *) &B2D2;
    swab(PTR, PTR, 2);
    RX.OP_RX.B2D2 = B2D2;
} else
{
    return (4);
}

SRX2:
printf("\noperc = %s ", TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAC);
printf("  BASE %d DELTA %d\n", NBASRG, DELTA);
RX.OP_RX.R1X2 = R1X2;
STXT(4);
return (0);
}

```

## **Выводы**

В рамках второго этапа курсовой работы по написанию компилятора с языка ассемблер были выполнены все поставленные задачи:

- 4) Расширена таблица машинных команд
- 5) Модифицированы обработчики команд DC и DS
- 6) Существующий компилятор доработан с учетом новых правил

В результате выполнения работы был получен объектный модуль. Проверить правильность его работы можно с помощью абсолютного загрузчика и эмулятора машины(третий этап курсовой работы). Замечания по коду сохраняются.

Минобрнауки России  
Санкт-Петербургский государственный политехнический университет  
Институт информационных технологий и управления  
**Кафедра «Информационные и управляющие системы»**

## **КУРСОВОЙ ПРОЕКТ**

Разработка учебной системы программирования

В а р и а н т 7

**Абсолютный загрузчик, эмулятор и отладчик**  
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 .....                     | 33 |
| Модификация кода загрузчика ..... | 33 |
| Таблица машинных операций .....   | 33 |
| Функции обработки операций.....   | 34 |
| Выводы .....                      | 36 |



## Задание

Воспользовавшись результатами второго этапа курсовой работы, доработать существующий загрузчик объектного представления программы. Произвести пошаговое выполнение программы, отслеживая состояние регистров и памяти. Это действие необходимо, чтобы убедиться в правильности второго этапа.

*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;
```

Необходимо доработать загрузчик, дополнив его новой функциональностью. В новую функциональность входит поддержка новых команд (LH, STH, SRL, SLL, OR, NR). Данные команды необходимо внести в таблицу машинных операций, а так же написать соответствующие обработчики.

## Модификация кода загрузчика

В рамках курсовой работы были проведены модификации и дополнения, которые позволяют решать поставленную в условии задачу.

## Таблица машинных операций

```
T_MOP [NOP] = { {
{{ 'B', 'A', 'L', 'R', ' ' } , '\x05' , 2 , FRR} ,
{{ 'B', 'C', 'R', ' ' , ' ' } , '\x07' , 2 , FRR} ,
{{ 'S', 'T', ' ' , ' ' , ' ' } , '\x50' , 4 , FRX} ,
{{ 'L', ' ' , ' ' , ' ' , ' ' } , '\x58' , 4 , FRX} ,
{{ 'A', ' ' , ' ' , ' ' , ' ' } , '\x5A' , 4 , FRX} ,
{{ 'S', ' ' , ' ' , ' ' , ' ' } , '\x5B' , 4 , FRX} ,
{{ 'L', 'H', ' ' , ' ' , ' ' } , '\x48' , 4 , FRX} ,
{{ 'S', 'R', 'L', ' ' , ' ' } , '\x01' , 4 , FRX} ,
{{ 'S', 'L', 'L', ' ' , ' ' } , '\x02' , 4 , FRX} ,
{{ 'O', 'R', ' ' , ' ' , ' ' } , '\x16' , 2 , FRR} ,
{{ 'N', 'R', ' ' , ' ' , ' ' } , '\x14' , 2 , FRR} ,
{{ 'S', 'T', 'H', ' ' , ' ' } , '\x40' , 4 , FRX} ,
};
```

Серым цветом выделены строки, соответствующие добавленным операциям.

## Функции обработки операций

### 1) Программные обработчики введенных команд

```
switch (T_MOP[k].CODOP{
    case '\x05' : P_BALR break
    case '\x07' : { i = P_BCR();
                   getch();
                   if (i == 1)
                       return 8;
                   }
                   break;
    case '\x50' : P_ST();
                   break;
    case '\x58' : P_L();
                   break;
    case '\x5A' : P_A();
                   break;
    case '\x5B' : P_S();
                   break;
    case '\x48' : P_LH();
                   break;
    case '\x01' : P_SRL();
                   break;
    case '\x02' : P_SLL();
                   break;
    case '\x16' : P_OR();
                   break;
    case '\x14' : P_NR();
                   break;
    case '\x40' : P_STH();
}
}
```

### 2) Функция обработки операции типа RX

```
int FRX(void)
{
    int i, j;

    for (i = 0; i < NOP; i++)
    {
        if (INST[0] == T_MOP[i].CODOP)
        {
            waddstr(wgreen, " ");
            for (j = 0; j < 5; j++)
                waddch(wgreen, T_MOP[i].MNCOP[j]);
            waddstr(wgreen, " ");

            j = INST[1] >> 4;
            R1 = j;
            wprintw(wgreen, "%.1d, ", j);

            j = INST[2] % 16;
            j = j * 256 + INST[3];
            D = j;
            wprintw(wgreen, "X'%.3X'(", j);
        }
    }
}
```

```

j = INST[1] % 16;
X = j;
wprintw(wgreen, "%1d, ", j);

j = INST[2] >> 4;
B = j;
wprintw(wgreen, "%1d)", j);

ADDR = VR[B] + VR[X] + D;
wprintw(wgreen, "%06lx\n", ADDR);
break;
}
}

```

### 3) Функция обработки операции SLL

```

int P_SLL()
{
    int sm = D;
    VR[R1] = VR[R1] << sm;
    return 0;
}

```

### 4) Функция обработки операции SRL

```

int P_SRL()
{
    int sm = D;
    VR[R1] = VR[R1] >> sm;
    return 0;
}

```

### 5) Функция обработки операции OR

```

int P_OR()
{
    VR[R1] = VR[R1] | VR[R2];
    return 0;
}

```

### 6) Функция обработки операции NR

```

int P_NR()
{
    VR[R1] = VR[R1] & VR[R2];
    return 0;
}

```

### 7) Функция обработки операции LH

```

int P_LH()
{
    int sm;
    ADDR = VR[B] + VR[X] + D;
    sm = (int)(ADDR - I);
    VR[R1] = OBLZ[BAS_IND + CUR_IND + sm] * 0x100L + OBLZ[BAS_IND +
CUR_IND + sm + 1];
    return 0;
}

```

### 8) Функция обработки операции STH

```

int P_STH()
{
    int sm,i;
    char bytes[2];

```

```

ADDR = VR[B] + VR[X] + D;
sm = (int) (ADDR - I);
bytes[0] = ((VR[R1] % 0x10000L) - ((VR[R1] % 0x10000L) % 0x100)) / 0x100;
bytes[1] = (VR[R1] % 0x10000L) % 0x100;
for (i=0; i<2; i++)
    OBLZ[BAS_IND + CUR_IND + sm + i] = bytes[i];
return 0;
}

```

## Выводы

В рамках проведенной работы был модифицирован абсолютный загрузчик и отладчик. Выполненные изменения позволили провести выполнение сформированного во втором этапе объектного представления:

- 7) Расширена таблица машинных команд
- 8) Модифицированы обработчики новых машинных команд
- 9) Осуществлена проверка работы модуля

В рамках отладочного запуска скомпилированной объектной карты была произведена проверка правильности результатов второго этапа. Результат соответствует требованиям задания.

Минобрнауки России  
Санкт-Петербургский государственный политехнический университет  
Институт информационных технологий и управления  
**Кафедра «Информационные и управляющие системы»**

## **КУРСОВОЙ ПРОЕКТ**

Разработка учебной системы программирования

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 .....                   | 39 |
| Технология.....                 | 39 |
| Эквивалент на Ассемблере .....  | 40 |
| Исходный код .....              | 41 |
| Лексический анализатор .....    | 41 |
| Синтаксический анализатор ..... | 43 |
| Выводы.....                     | 47 |

## 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 *rzt, char *lit);
void odr(char *tpe, char *rzt);
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 '(' rzt ')' INIT '(' lit ')' ';' { odi($2, $6, $10);
}
    | DCL ipe BIT '(' rzt ')' INIT '(' lib ')' ';' { odi($2, $5, $9); }
    ;
odr:  DCL ipe BIN FIXED '(' rzt ')' ';' { odr($2, $6); }
    | DCL ipe BIT '(' rzt ')' ';' { odr ($2, $5); }
    ;
ipe: IDENT {$$=$1;}
    ;
rzt: NUM {$$=$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 *rzzr) {

    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(rzzr, "31", 2)) {
        s1[15]='F';
    } else if (!memcmp (rzzr, "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 identificador ");
        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».