Минобрнауки России

Санкт-Петербургский государственный политехнический университет

Институт информационных технологий и управления

**Кафедра «Информационные и управляющие системы»**

**КУРСОВОЙ ПРОЕКТ**

Разработка учебной системы программирования

Вариант 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

Вариант 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 3](#_Toc356981358)

[План работы 3](#_Toc356981359)

[Эквивалент на Ассемблере 4](#_Toc356981360)

[Модификация грамматики языка 5](#_Toc356981361)

[Матрица смежности 9](#_Toc356981362)

[Таблица синтаксических правил 10](#_Toc356981363)

[Таблица входов в правила 12](#_Toc356981364)

[Модификация функций компилятора 13](#_Toc356981365)

[Выводы 19](#_Toc356981366)

# 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 функциональность. 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 В в регистр

LH 4,A Загрузка переменной А 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><ZNKL><IPE> | <SSTR> | (<LVI>)

15. <ZNK> ::= + | -

15.1 <ZNKL> ::= !!

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",
* "<ZNKL>" - 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>
29. DCL\_<IPE>\_BIN\_FIXED(<RZR>); -> <ODC>
30. DCL\_<IPE>\_BIN\_FIXED(<RZR>)INIT(<LIT>); -> <ODC>
31. DCL\_<IPE>\_BIT(<RZR>); -> <ODC>
32. DCL\_<IPE>\_BIT(<RZR>)INIT(<LIT>); -> <ODC>
33. <IDE> -> <IPE>
34. <CIF> -> <RZR>
35. <RZR><CIF> -> <RZR>
36. <MAN>B -> <LIT>
37. 1 -> <MAN>
38. <MAN>0 -> <MAN>
39. <MAN>1 -> <MAN>
40. <IPE>=<AVI>; -> <OPA>
41. <IPE>=<LVI>; -> <OPL>
42. <LIT> -> <AVI>
43. <IPE> -> <AVI>
44. <AVI><ZNK><LIT> -> <AVI>
45. <AVI><ZNK><IPE> -> <AVI>
46. <IPE> -> <LVI>
47. <LVI><ZNKL><IPE> -> <LVI>
48. <SSTR> -> <LVI>
49. (<LVI>) -> <LVI>
50. + -> <ZNK>
51. - -> <ZNK>
52. !! -> <ZNKL>
53. SUBSTR(<LVI>, <RZR>, <RZR>) -> <SSTR>
54. 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>

└-> <CIF> -> <IDE>

└-> <IPR>

└-> <IPE>

1. <BUK> -> <IDE>
2. A -> <BUK>
3. B -> <BUK>
4. C -> <BUK>
5. D -> CL\_<IPE>\_BIN\_FIXED(<RZR>)INIT(<LIT>); -> <ODC>

| | └-> ; -> <ODC>

| └->T(<RZR>)INIT(<LIB>); -> <ODC>

| └-> ; -> <ODC>

└ <BUK>

1. E -> ND\_<IPR> -> <OEN>

└-> <BUK>

1. M -> <BUK>
2. P -> <BUK>
3. X -> <BUK>
4. 0 -> <CIF>
5. 1 -> <CIF>

└-> <MAN>

1. 2 -> <CIF>
2. 3 -> <CIF>
3. 4 -> <CIF>
4. 5 -> <CIF>
5. 6 -> <CIF>
6. 7 -> <CIF>
7. 8 -> <CIF>
8. 9 -> <CIF>
9. <ODC> -> <TEL>
10. <TEL> -> <ODC> -> <TEL>

└-> <OPA> -> <TEL>

└-> <OPL> -> <TEL>

1. <CIF> -> <RZR>
2. <RZR><CIF> -> <RZR>
3. <MAN> -> B -> <LIT>

└-> 0 -> <MAN>

└-> 1 -> <MAN>

1. <IPE> -> =<AVI>; -> <OPA>

| └-> <LVI>; -> <OPL>

└-> <AVI>

└-> <LVI>

1. <LIT> -> <AVI>
2. <AVI><ZNK> -> <LIT> -> <AVI>

└-> <IPE> -> <AVI>

1. <LVI><ZNKL><IPE> -> <LVI>
2. <SSTR> -> <LVI>
3. (<LVI>) -> <LVI>
4. + -> <ZNK>
5. - -> <ZNK>
6. !! -> <ZNKL>
7. SUBSTR(<LVI>, <RZR>, <RZR>) -> <SSTR>

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:IPE:IPR:LIT:MAN:ODC:OEN:OPA:OPR:PRO:RZR:TEL:ZNK:OPL:LVI:ZKL:SUS|

|\_\_\_\_\_\_\_\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_:\_\_\_|

{/\*IPE\*/ 1 , 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 },

{/\*LVI\*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 },

{/\*ZNKL\*/ 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 , 1 , 0 , 0 },

{/\* S\*/ 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 , 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 , 1 , 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, введённые 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';

}

1. 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;

}

1. 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;

}

1. 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, "Naznacit' registr bazoy", 23);

ZKARD();

return 0;

}

1. 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 programmy", 15);

ZKARD();

return 0;

}

1. 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;

}

1. 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, "Formirovanye znacenya logic.virazenya", 37);

ZKARD();

return 0;

}

else

return 3;

}

}

return 4;

}

1. Функция логического выражения на первом проходе

int LVI1() {

return 0;

}

1. 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, "Formirovanye promezutocnogo znacenya",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;

}

}

1. Функция логического знака на первом проходе

int ZKL1() {

return 0;

}

1. Функция логического знака на втором проходе

int ZKL2() {

return 0;

}

1. Функция вычисления SUS на первом проходе (оп. substr)

int SUS1() {

return 0;

}

1. 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, "Sdvig 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, "Logiceskoye 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

1. Большое количество операторов goto
2. 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

Вариант 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 3](#_Toc356983526)

[План работы 4](#_Toc356983527)

[Расширение таблицы команд 4](#_Toc356983528)

[Модификация кода компилятора 5](#_Toc356983529)

[Первый просмотр: 5](#_Toc356983530)

[Второй просмотр: 7](#_Toc356983531)

[Выводы 10](#_Toc356983532)

# 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 В в регистр

LH 4,A Загрузка переменной А 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:

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

Для решения этой задачи необходимо:

1. расширить таблицу машинных команд;
2. 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].BXPROG = SRR;

T\_MOP[1].BXPROG = SRR;

T\_MOP[2].BXPROG = SRX;

T\_MOP[3].BXPROG = SRX;

T\_MOP[4].BXPROG = SRX;

T\_MOP[5].BXPROG = SRX;

T\_MOP[6].BXPROG = SRX;

T\_MOP[7].BXPROG = SRX;

T\_MOP[8].BXPROG = SRX;

T\_MOP[9].BXPROG = SRR;

T\_MOP[10].BXPROG = SRR;

T\_MOP[11].BXPROG = 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);

}

}

1. 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);

}

1. 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);

}

1. 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);

}

1. 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 = 0xfff - 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 > 0xfff)

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:

1. Расширена таблица машинных команд
2. Модифицированы обработчики команд DC и DS
3. 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

Вариант 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 3](#_Toc357546198)

[Модификация кода загрузчика 3](#_Toc357546199)

[Таблица машинных операций 3](#_Toc357546200)

[Функции обработки операций 4](#_Toc357546201)

[Выводы 6](#_Toc357546202)

# 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();

}

1. 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;

}

}

1. Функция обработки операции SLL

int P\_SLL()

{

int sm = D;

VR[R1] = VR[R1] << sm;

return 0;

}

1. Функция обработки операции SRL

int P\_SRL()

{

int sm = D;

VR[R1] = VR[R1] >> sm;

return 0;

}

1. Функция обработки операции OR

int P\_OR()

{

VR[R1] = VR[R1] | VR[R2];

return 0;

}

1. Функция обработки операции NR

int P\_NR()

{

VR[R1] = VR[R1] & VR[R2];

return 0;

}

1. 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;

}

1. 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:

1. Расширена таблица машинных команд
2. Модифицированы обработчики новых машинных команд
3. Осуществлена проверка работы модуля

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