

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

КУРСОВАЯ РАБОТА

Разработка учебной системы программирования
по дисциплине «Системы программирования»

Выполнили
студенты гр.53504/10

Руководитель
доцент



Е.А.Белокопытова
Д.А.Морозов

В.Я. Расторгуев

«__» _____ 2014 г.

Санкт-Петербург
2014

Содержание

Введение.....	3
1. Компилятор с языка высокого уровня PL1.....	6
1.1 Описание примера из личного задания	6
1.2 Постановка задачи.....	6
1.3 Перечень и пояснения модификаций состава и структуры БД компилятора.....	8
1.3.1 Изменения в синтаксисе анализатора.....	8
1.3.2 Изменения в таблице продукции	11
1.3.3 Изменения в таблице входов.....	13
1.3.4 Изменения в матрице смежности.....	13
1.4 Перечень и пояснения модификаций алгоритма программы-макета компилятора....	14
1.5 Результат работы обновленном макета компилятора с ЯВУ с вариантом из рабочего примера.....	19
1.6 Выводы.....	19
2. Компилятор с языка ассемблер.....	21
2.1 Описание примера из личного задания.....	21
2.2 Постановка задачи.....	21
2.3 Перечень и пояснения модификаций состава и структуры компилятора с языка Ассемблер.....	25
2.3.1 Добавление строк в таблицу машинных операций.....	25
2.3.2 Изменение блока подпрограмм, используемых при 1-ом просмотре.....	25
2.3.2 Изменение блока подпрограмм, используемых при 2-ом просмотре.....	27
2.4 Результат работы обновленном макета компилятора с ЯВУ с вариантом из рабочего примера.....	28
Выводы.....	30
3. Абсолютный загрузчик.....	32
3.1 Описание примера из личного задания.....	32
3.2 Постановка задачи.....	32
3.3 Изменения, внесенные в программу-макет эмулятора системы IBM 370.....	35
3.3 Результаты.....	37
Выводы.....	37
Заключение	39
Список литературы.....	40

Список рисунков

Рисунок 1.Программа-макет эмулятора и отладчика.....	4
Рисунок 2.Объектное представление исходного кода.....	23
Рисунок 3.Объектное представление исходного кода.....	24
Рисунок 4.Объектное представление исходного кода.....	29
Рисунок 5.Объектное представление исходного кода.....	30
Рисунок 6.Объектное представление исходного кода.....	33
Рисунок 7.Объектное представление исходного кода.....	34
Рисунок 8.Программа-макет эмулятора и отладчика.....	34
Рисунок 9.Программа-макет эмулятора и отладчика.....	37

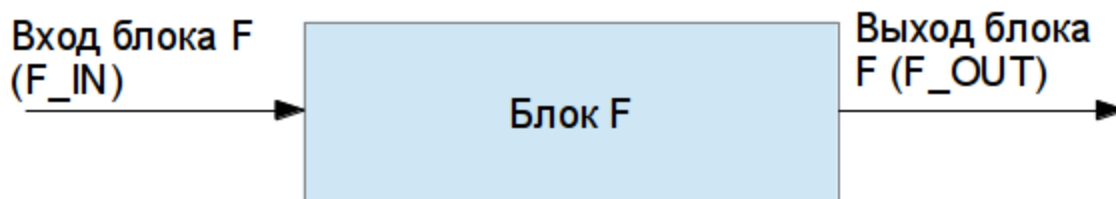
Введение.

Данная работа посвящена ознакомлению с подходами к построению систем программирования с использованием методик компиляции и трансляции кода, модульного построение, кросс-технологий.

В ходе работы будет реализована и дополнена существующая система программирования.

Постановка задачи

Необходимо выполнить доработку элементов макета учебной системы программирования (блок F на схеме) до уровня, позволяющего обрабатывать “новые” для макета конструкции языка высокого уровня, примененные в соответствующем варианте, получая на выходе результат работы программы макета в эмуляторе.



Вход (F_IN): код на языке высокого уровня PL/1

```
EX02: PROC OPTIONS ( MAIN );  
DCL A BIT ( 16 ) INIT ( '101'B );  
DCL B BIT ( 16 ) INIT ( '1010'B );  
DCL C BIT ( 16 ) INIT ( '111'B );  
DCL D BIT ( 16 );  
D = ( A ! B ) & C;  
END EX02;
```

Выход (F_OUT): результат работы программы макета в эмуляторе.

					R00: 00000000
					R01: 00000000
					R02: 0004C1A2
					R03: 0000A000
					R04: 0000E000
					R05: 00000000
					R06: 00000000
					R07: 00000000
804C1A0:	0520	BALR	2, 0		R08: 00000000
804C1A2:	48302018	LH	3, X'018'(0, 2)	804C1BA	R09: 00000000
804C1A6:	4840201A	LH	4, X'01A'(0, 2)	804C1BC	R10: 00000000
804C1AA:	1634	OR	3, 4		R11: 00000000
804C1AC:	4840201C	LH	4, X'01C'(0, 2)	804C1BE	R12: 00000000
804C1B0:	1434	NR	3, 4		R13: 00000000
804C1B2:	4830201E	STH	3, X'01E'(0, 2)	804C1C0	R14: 00000000
804C1B6:	07FE	BCR	15, 14		R15: 00000000
804C1B8:	0000A000	A000E000	A0000000 00000000 /* */	
804C1C8:	00000000	00000000	00000000 00000000 /* */	
804C1D8:	00000000	00000000	00000000 00000000 /* */	
804C1E8:	00000000	00000000	00000000 00000000 /* */	
804C1F8:	00000000	00000000	00000000 00000000 /* */	
804C208:	00000000	00000000	00000000 00000000 /* */	
804C218:	00000000	00000000	00000000 00000000 /* */	
804C228:	00000000	00000000	00000000 00000000 /* */	
roToBHotb K BbInoJIHeHul0 o4epeghou Komahgbl c aghpecoM 804C1B6					
"PgUp","PgDn","Up","Down"→npocMoTp gamma: "Enter"→BbInoJIHuTb o4epeghul0 Komah					

Рисунок 1. Программа-макет эмулятора и отладчика

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

КУРСОВАЯ РАБОТА

**Разработка учебной системы программирования.
Компилятор с языка высокого уровня
по дисциплине «Системы программирования»**

Выполнили
студенты гр.53504/10

Руководитель
доцент



Е.А.Белокопытова
Д.А.Морозов

В.Я. Расторгуев

«___» _____ 2014 г.

Санкт-Петербург
2014

1. Компилятор с языка высокого уровня PL1

1.1 Описание примера из личного задания

Данная часть курсового проекта имеет своей целью получение практических навыков построения компилятора с языка высокого уровня (ЯВУ). В качестве языка высокого уровня выбран язык, образованный из подмножества языковых конструкций ПЛ1, а исходная программа готовится в виде текстового файла технологической ЭВМ с расширением *.pli.

В данном курсовом проекте рассматривается реализация компилятора для следующей исходной программы:

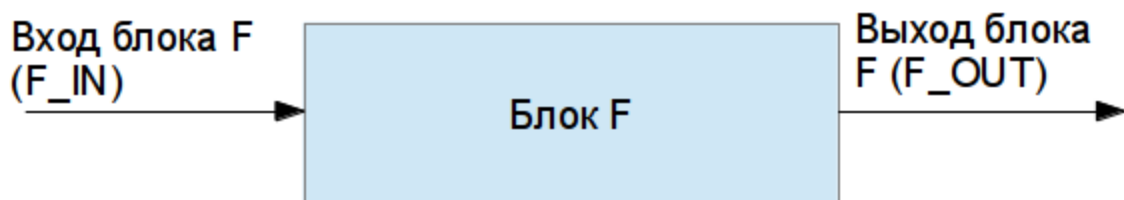
Вариант N 2

```
EX02: PROC OPTIONS ( MAIN );  
DCL A BIT ( 16 ) INIT ( '101'B );  
DCL B BIT ( 16 ) INIT ( '1010'B );  
DCL C BIT ( 16 ) INIT ( '111'B );  
DCL D BIT ( 16 );  
D = ( A ! B ) & C;  
END EX02;
```

Ассемблеровский эквивалент исходной программы формируется в виде текстового файла технологической ЭВМ с расширением *.ass .

1.2 Постановка задачи

Необходимо выполнить доработку элементов макета учебной системы программирования (блока F на схеме) в части компилятора с ЯВУ до уровня, позволяющего обрабатывать “новые” для макета конструкции языка высокого уровня, примененные в соответствующем варианте, получая на выходе ассемблеровский код.



Вход (F_IN): код на языке высокого уровня PL/1:

```
EX02: PROC OPTIONS ( MAIN );
```

```

DCL A BIT ( 16 ) INIT ( '101'B );
DCL B BIT ( 16 ) INIT ( '1010'B );
DCL C BIT ( 16 ) INIT ( '111'B );
DCL D BIT ( 16 );
D = ( A ! B ) & C;
END EX02;

```

Выход (F_OUT): код на языке Ассемблер:

Метка	КОП	Операнды	Комментарии
EX02	START	0	Начало программы
	BALR	RBASE, 0	Загрузить регистр базы
	USING	*, RBASE	Назначить регистр базой
	LH	R1, A	Загрузка переменной в регистр
	LH	R2, B	Загрузка переменной в регистр
	OR	R1, R2	Формирование промежуточного значения
	LH	R2, C	Загрузка переменной в регистр
	NR	R1, R2	Формирование промежуточного значения
	STH	R1, D	Формирование значения арифметического выражения
	BCR	15, 14	Выход из программы
	DS	0H	Выравнивание на 2 байта
A	DC	BL2'101'	Определение переменной
B	DC	BL2'1010'	Определение переменной
C	DC	BL2'111'	Определение переменной
D	DS	BL2	Определение переменной
RBASE	EQU	2	
R1	EQU	3	
R2	EQU	4	
	END	EX02	Конец программы

Для работы данного примера на языке высокого уровня в программе-макете должны быть произведены следующие изменения:

- Добавлена поддержка переменных типа “битовые строки” (с возможностью начальной инициализации и без).
- Реализована возможность работы с поразрядными логическими выражениями, выполнение поразрядных логических операций над переменными.

В качестве особенностей выполнения работы с преподавателем согласовано:

- Порядок выполнения логических операций, определяемый скобками игнорируется – операции равнозначны.
- Логические переменные длиной в два байта выравниваются в памяти по левому краю
- Максимальная длина данного типа BIT – 16 бит (2 байта)

1.3 Перечень и пояснения модификаций состава и структуры БД компилятора

1.3.1 Изменения в синтаксисе анализатора

Для изменения была выбрана версия программы-макета под ОС Linux.

Далее приведен список дополнений синтаксиса для базы данных компилятора, жирным шрифтом выделены отличия от БД программы-макета:

1. `<ODC> ::= DCL_<IPE>_BIN_FIXED(<RZR>)INIT(<LIT>); |
DCL_<IPE>_BIN_FIXED(<RZR>); |
DCL_<IPE>_BIT(<RZR>)INIT(<LIB>); |
DCL_<IPE>_BIT(<RZR>);`
2. `<LIB> ::= '<STR>'B`
3. `<STR> ::= 0 | 1 | <STR>0 | <STR>1`
4. `<OPA> ::= <IPE>=<AVI>; |
<IPE>=<LVI>;`
5. `<LVI> ::= <IPE> | <LVI><LOP><IPE>`
6. `<LOP> ::= ! | &`

Здесь использованы следующие метасимволы и символы:

- "<LIB>" - нетерминал "литерал для типа bit",
- "<STR>" - нетерминал "строка",
- "<LVI>" - нетерминал "логическое выражение",
- "<LOP>" - нетерминал "логическая операция",

Так как грамматика языка примера, приведенная выше и заданная в нотациях метаязыка Бэкуса, непосредственно алгоритмом синтаксического анализа использована быть не может, то ее следует привести к виду, адекватному для машинного использования. Для этого сделаем несколько последовательных преобразований формы представления грамматики, имея конечную цель – получение грамматики в виде некоторого списка, который при программировании на языке “С” может быть представлен как массив структур.

Сначала перепишем каждое правило грамматики, заменяя формат метаязыка Бэкуса на формат продукций (или на форму распознавания):

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. $C \rightarrow \langle \text{BUK} \rangle$
10. $D \rightarrow \langle \text{BUK} \rangle$
11. $E \rightarrow \langle \text{BUK} \rangle$
12. $M \rightarrow \langle \text{BUK} \rangle$
13. $P \rightarrow \langle \text{BUK} \rangle$
14. $X \rightarrow \langle \text{BUK} \rangle$
15. $0 \rightarrow \langle \text{CIF} \rangle$
16. $1 \rightarrow \langle \text{CIF} \rangle$
17. $2 \rightarrow \langle \text{CIF} \rangle$
18. $3 \rightarrow \langle \text{CIF} \rangle$
19. $4 \rightarrow \langle \text{CIF} \rangle$
20. $5 \rightarrow \langle \text{CIF} \rangle$
21. $6 \rightarrow \langle \text{CIF} \rangle$
22. $7 \rightarrow \langle \text{CIF} \rangle$
23. $8 \rightarrow \langle \text{CIF} \rangle$
24. $9 \rightarrow \langle \text{CIF} \rangle$
25. $\langle \text{ODC} \rangle \rightarrow \langle \text{TEL} \rangle$
26. $\langle \text{TEL} \rangle \langle \text{ODC} \rangle \rightarrow \langle \text{TEL} \rangle$
27. $\langle \text{TEL} \rangle \langle \text{OPA} \rangle \rightarrow \langle \text{TEL} \rangle$
28. $\text{DCL_} \langle \text{IPE} \rangle _ \text{BIN_FIXED}(\langle \text{RZR} \rangle); \rightarrow \langle \text{ODC} \rangle$
29. $\text{DCL_} \langle \text{IPE} \rangle _ \text{BIN_FIXED}(\langle \text{RZR} \rangle) \text{INIT}(\langle \text{LIT} \rangle); \rightarrow \langle \text{ODC} \rangle$
30. $\text{DCL_} \langle \text{IPE} \rangle _ \text{BIT}(\langle \text{RZR} \rangle); \rightarrow \langle \text{ODC} \rangle$
31. $\text{DCL_} \langle \text{IPE} \rangle _ \text{BIT}(\langle \text{RZR} \rangle) \text{INIT}(\langle \text{LIB} \rangle); \rightarrow \langle \text{ODC} \rangle$
32. $\langle \text{IDE} \rangle \rightarrow \langle \text{IPE} \rangle$
33. $\langle \text{CIF} \rangle \rightarrow \langle \text{RZR} \rangle$
34. $\langle \text{RZR} \rangle \langle \text{CIF} \rangle \rightarrow \langle \text{RZR} \rangle$
35. $\langle \text{MAN} \rangle \text{B} \rightarrow \langle \text{LIT} \rangle$
36. $\langle \text{STR} \rangle \text{B} \rightarrow \langle \text{LIB} \rangle$
37. $1 \rightarrow \langle \text{MAN} \rangle$
38. $\langle \text{MAN} \rangle 0 \rightarrow \langle \text{MAN} \rangle$
39. $\langle \text{MAN} \rangle 1 \rightarrow \langle \text{MAN} \rangle$
40. $0 \rightarrow \langle \text{STR} \rangle$
41. $1 \rightarrow \langle \text{STR} \rangle$
42. $\langle \text{STR} \rangle 0 \rightarrow \langle \text{STR} \rangle$
43. $\langle \text{STR} \rangle 1 \rightarrow \langle \text{STR} \rangle$
44. $\langle \text{IPE} \rangle = \langle \text{AVI} \rangle; \rightarrow \langle \text{OPA} \rangle$
45. $\langle \text{IPE} \rangle = \langle \text{LVI} \rangle; \rightarrow \langle \text{OPA} \rangle$
46. $\langle \text{LIT} \rangle \rightarrow \langle \text{AVI} \rangle$
47. $\langle \text{IPE} \rangle \rightarrow \langle \text{AVI} \rangle$
48. $\langle \text{AVI} \rangle \langle \text{ZNK} \rangle \langle \text{LIT} \rangle \rightarrow \langle \text{AVI} \rangle$
49. $\langle \text{AVI} \rangle \langle \text{ZNK} \rangle \langle \text{IPE} \rangle \rightarrow \langle \text{AVI} \rangle$
50. $\langle \text{IPE} \rangle \rightarrow \langle \text{LVI} \rangle$
51. $\langle \text{LVI} \rangle \langle \text{LOP} \rangle \langle \text{IPE} \rangle \rightarrow \langle \text{LVI} \rangle$
52. $+ \rightarrow \langle \text{ZNK} \rangle$
53. $- \rightarrow \langle \text{ZNK} \rangle$
54. $! \rightarrow \langle \text{LOP} \rangle$

55. & → <LOP>

56. 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. 8<OPR><TEL><OEN> → <PRO>

2. <IPR>:PROC_OPTIONS(MAIN); → <OPR>

3. <IDE> → <BUK> → <IDE>
 └→ <CIF> → <IDE>
 └→ <IPR>
 └→ <IPE>

4. <BUK> → <IDE>

5. A → <BUK>

6. B → <BUK>

7. C → <BUK>

8. D → CL_<IPE>_BIN_FIXED(<RZR>)_INIT(<LIT>); → <ODC>
 └→ T(<RZR>)_INIT(<LIB>); → <ODC>
 └→ ; → <ODC>
 └→ <BUK>

9. E → ND_<IPR>; → <OEN>
 └→ <BUK>

10. M → <BUK>

11. P → <BUK>

12. X → <BUK>

13. 0 → <CIF>
 └→ <STR>

14. 1 → <CIF>
 └→ <MAN>
 └→ <STR>

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>
 └→ <OPA> → <TEL>

25. <CIF> → <RZR>

26. <RZR><CIF> → <RZR>

27. <MAN> → B → <LIT>
 └→ 0 → <MAN>

$$L \rightarrow 1 \rightarrow \langle \text{MAN} \rangle$$

28. ' $\langle \text{STR} \rangle$ 'B \rightarrow $\langle \text{LIB} \rangle$

29. $\langle \text{STR} \rangle \rightarrow 0 \rightarrow \langle \text{STR} \rangle$

$$\mathbb{L} \rightarrow 1 \rightarrow \langle \text{STR} \rangle$$

30. $\langle \text{IPE} \rangle \rightarrow = \langle \text{AVI} \rangle; \rightarrow \langle \text{OPA} \rangle$

$$| \quad \mathbb{L} \rightarrow \langle \text{LVI} \rangle; \rightarrow \langle \text{OPA} \rangle$$
$$L \rightarrow \langle AVI \rangle$$
$$\mathbf{L} \rightarrow \langle \mathbf{LVI} \rangle$$

31. $\langle \text{LIT} \rangle \rightarrow \langle \text{AVI} \rangle$

32. $\langle \text{AVI} \rangle \langle \text{ZNK} \rangle \rightarrow \langle \text{LIT} \rangle \rightarrow \langle \text{AVI} \rangle$

$$\mathbf{L} \rightarrow \langle \text{IPE} \rangle \rightarrow \langle \text{AVI} \rangle$$

33. $\langle \text{LVI} \rangle \langle \text{LOP} \rangle \langle \text{IPE} \rangle \rightarrow \langle \text{LVI} \rangle$

34. $+ \rightarrow \langle \text{ZNK} \rangle$

35. $\rightarrow \langle \text{ZNK} \rangle$

36. ! \rightarrow <LOP>

37. & → <LOP>

```

struct
{
    int  POSL;
    int  PRED;
    char DER[4];
    int  ALT;
} SINT [ NSINT ] =
/*
|  NN      :      посл      :      пред      :      дер      :      альт
|  _____:_____ : _____:_____ : _____:_____ | */
/*      ВХОД С СИМВОЛА - D      */
{ /*.      52      .*/      53      ,      0      ,      "D      "      ,      0      },
{ /*.      53      .*/      54      ,      52      ,      "C      "      ,      100      },
{ /*.      54      .*/      55      ,      53      ,      "L      "      ,      0      },
{ /*.      55      .*/      56      ,      54      ,      "      "      ,      0      },
{ /*.      56      .*/      57      ,      55      ,      "IPE"      ,      0      },
{ /*.      57      .*/      58      ,      56      ,      "      "      ,      0      },
{ /*.      58      .*/      59      ,      57      ,      "B      "      ,      0      },
{ /*.      59      .*/      60      ,      58      ,      "I      "      ,      0      },
{ /*.      60      .*/      61      ,      59      ,      "N      "      ,      83      },
{ /*.      61      .*/      62      ,      60      ,      "      "      ,      0      },
{ /*.      62      .*/      63      ,      61      ,      "F      "      ,      0      },
{ /*.      63      .*/      64      ,      62      ,      "I      "      ,      0      },
{ /*.      64      .*/      65      ,      63      ,      "X      "      ,      0      },
{ /*.      65      .*/      66      ,      64      ,      "E      "      ,      0      },
{ /*.      66      .*/      67      ,      65      ,      "D      "      ,      0      },
{ /*.      67      .*/      68      ,      66      ,      "(      "      ,      0      },
{ /*.      68      .*/      69      ,      67      ,      "RZR"      ,      0      },
{ /*.      69      .*/      70      ,      68      ,      ")      "      ,      0      },
{ /*.      70      .*/      71      ,      69      ,      "I      "      ,      80      },

```

```

{/*. 71 .*/ 72 , 70 , "N " , 0 },
{/*. 72 .*/ 73 , 71 , "I " , 0 },
{/*. 73 .*/ 74 , 72 , "T " , 0 },
{/*. 74 .*/ 75 , 73 , "( " , 0 },
{/*. 75 .*/ 76 , 74 , "LIT" , 0 },
{/*. 76 .*/ 77 , 75 , ") " , 0 },
{/*. 77 .*/ 78 , 76 , ";" " , 0 },
{/*. 78 .*/ 79 , 77 , "ODC" , 0 },
{/*. 79 .*/ 0 , 78 , "*" " , 0 },
{/*. 80 .*/ 81 , 69 , ";" " , 0 },
{/*. 81 .*/ 82 , 80 , "ODC" , 0 },
{/*. 82 .*/ 0 , 81 , "*" " , 0 },
{/*. 83 .*/ 84 , 59 , "T " , 0 },
{/*. 84 .*/ 85 , 83 , "( " , 0 },
{/*. 85 .*/ 86 , 84 , "RZR" , 0 },
{/*. 86 .*/ 87 , 85 , ") " , 0 },
{/*. 87 .*/ 88 , 86 , "I " , 97 },
{/*. 88 .*/ 89 , 87 , "N " , 0 },
{/*. 89 .*/ 90 , 88 , "I " , 0 },
{/*. 90 .*/ 91 , 89 , "T " , 0 },
{/*. 91 .*/ 92 , 90 , "( " , 0 },
{/*. 92 .*/ 93 , 91 , "LIB" , 0 },
{/*. 93 .*/ 94 , 92 , ") " , 0 },
{/*. 94 .*/ 95 , 93 , ";" " , 0 },
{/*. 95 .*/ 96 , 94 , "ODC" , 0 },
{/*. 96 .*/ 0 , 95 , "*" " , 0 },
{/*. 97 .*/ 98 , 86 , ";" " , 0 },
{/*. 98 .*/ 99 , 97 , "ODC" , 0 },
{/*. 99 .*/ 0 , 98 , "*" " , 0 },
{/*. 100 .*/ 101 , 52 , "BUK" , 0 },
{/*. 101 .*/ 0 , 100 , "*" " , 0 },
/* вход с символа - 0 */
{/*. 121 .*/ 122 , 0 , "0 " , 0 },
{/*. 122 .*/ 123 , 121 , "CIF" , 124 },
{/*. 123 .*/ 0 , 122 , "*" " , 0 },
{/*. 124 .*/ 125 , 121 , "STR" , 0 },
{/*. 125 .*/ 0 , 124 , "*" " , 0 },
/* вход с символа - 1 */
{/*. 126 .*/ 127 , 0 , "1 " , 0 },
{/*. 127 .*/ 128 , 126 , "CIF" , 129 },
{/*. 128 .*/ 0 , 127 , "*" " , 0 },
{/*. 129 .*/ 130 , 126 , "MAN" , 131 },
{/*. 130 .*/ 0 , 129 , "*" " , 0 },
{/*. 131 .*/ 132 , 126 , "STR" , 0 },
{/*. 132 .*/ 0 , 131 , "*" " , 0 },
/* вход с символа - ' */
{/*. 184 .*/ 185 , 0 , "'" " , 0 },
{/*. 185 .*/ 186 , 184 , "STR" , 0 },
{/*. 186 .*/ 187 , 185 , "'" " , 0 },
{/*. 187 .*/ 188 , 186 , "B " , 0 },
{/*. 188 .*/ 189 , 187 , "LIB" , 0 },
{/*. 189 .*/ 0 , 188 , "*" " , 0 },
/* вход с символа - STR */
{/*. 190 .*/ 191 , 0 , "STR" , 0 },
{/*. 191 .*/ 192 , 190 , "0 " , 194 },
{/*. 192 .*/ 193 , 191 , "STR" , 0 },
{/*. 193 .*/ 0 , 192 , "*" " , 0 },
{/*. 194 .*/ 195 , 190 , "1 " , 0 },
{/*. 195 .*/ 196 , 194 , "STR" , 0 },
{/*. 196 .*/ 0 , 195 , "*" " , 0 },
/* вход с символа - IPE */
{/*. 197 .*/ 198 , 0 , "IPE" , 0 },
{/*. 198 .*/ 199 , 197 , "=" " , 207 },
{/*. 199 .*/ 200 , 198 , "AVI" , 203 },

```

```

{/*. 200      .*/ 201 , 199 , ";" , 0 },
{/*. 201      .*/ 202 , 200 , "OPA" , 0 },
{/*. 202      .*/ 0 , 201 , "*" , 0 },
{/*. 203      .*/ 204 , 198 , "LVI" , 0 },
{/*. 204      .*/ 205 , 203 , ";" , 0 },
{/*. 205      .*/ 206 , 204 , "OPA" , 0 },
{/*. 206      .*/ 0 , 205 , "*" , 0 },
{/*. 207      .*/ 208 , 197 , "AVI" , 209 },
{/*. 208      .*/ 0 , 207 , "*" , 0 },
{/*. 209      .*/ 210 , 197 , "LVI" , 0 },
{/*. 210      .*/ 0 , 209 , "*" , 0 },
/*      вход с символа - LVI      */
{/*. 222      .*/ 223 , 0 , "LVI" , 0 },
{/*. 223      .*/ 224 , 222 , "LOP" , 0 },
{/*. 224      .*/ 225 , 223 , "IPE" , 0 },
{/*. 225      .*/ 226 , 224 , "LVI" , 0 },
{/*. 226      .*/ 0 , 225 , "*" , 0 },
/*      вход с символа - !      */
{/*. 233      .*/ 234 , 0 , "!" , 0 },
{/*. 234      .*/ 235 , 233 , "LOP" , 0 },
{/*. 235      .*/ 0 , 234 , "*" , 0 },
/*      вход с символа - &      */
{/*. 236      .*/ 237 , 0 , "&" , 0 },
{/*. 237      .*/ 238 , 236 , "LOP" , 0 },
{/*. 238      .*/ 0 , 237 , "*" , 0 }
}

```

1.3.3 Изменения в таблице входов

В таблицу входов в кусты были добавлены записи для вновь добавленных нетерминалов и терминалов:

```

struct
{
    char SYM [4];
    int  VX;
    char TYP;
} VXOD [ NVXOD ] =
.....
{/*. 7      .*/ "LIB" , 0 , 'N' },
{/*. 9      .*/ "LOP" , 0 , 'N' },
{/*. 10     .*/ "LVI" , 222 , 'N' },
{/*. 18     .*/ "STR" , 190 , 'N' },
{/*. 50     .*/ "!" , 233 , 'T' },
{/*. 51     .*/ "&" , 236 , 'T' },
{/*. 58     .*/ "'" , 184 , 'T' },
.....
)

```

1.3.4 Изменения в матрице смежности

Изменение таблицы матрицы смежности:

```

char TPR [ NVXOD ] [ NNETRM ] =
{
/*
| AVI:BUK:CIF:IDE:IPE:IPR:LIB:LIT:LOP:LVI:MAN:ODC:OEN:OPA:OPR:PRO:RZR:STR:TEL:ZNK|
| : : : : : : : : : : : : : : : : : : : : | */
{/*.AVI*/ 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 },
{/*.BUK*/ 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 },
{/*.CIF*/ 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 },
{/*.IDE*/ 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 },

```

```

{ /* IPE */ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 },
{ /* IPR */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 },
{ /* LIB */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* LIT */ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* LOP */ 0, 0, 0, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* MAN */ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* ODC */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },
{ /* OEN */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* OPA */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* OPR */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 },
{ /* PRO */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* RZR */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },
{ /* STR */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },
{ /* TEL */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },
{ /* ZNK */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* A */ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* B */ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* C */ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* D */ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* E */ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* F */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* H */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* I */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* L */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* M */ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* N */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* O */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* P */ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* R */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* S */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* T */ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* X */ 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 },
{ /* 1 */ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },
{ /* 2 */ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* 3 */ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* 4 */ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* 5 */ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* 6 */ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* 7 */ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* 8 */ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ /* 9 */ 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, 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, 0, 1, 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, 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, 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, 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, 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, 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 }
};

```

1.4 Перечень и пояснения модификаций алгоритма программы-макета компилятора

В процессе изменения программы-макеты были сделаны следующие изменения:

1. На основе дополненного синтаксиса базы данных базы данных компилятора были заново заполнены следующие таблицы:

- таблица синтаксических правил, записанных в форме распознавания;
- таблица входов в «кусты» (корней) грамматических правил;
- таблица матрицы смежности.

2. **Лексический анализатор:** в функции compress_ISXTXT() добавлено

распознавание символов "!" и "&".

```
void compress_ISXTXT () {
....
if ((ISXTXT [ I1 ][ I2 ] == '+' ||
      ISXTXT [ I1 ][ I2 ] == '-' || /* в случае встречи одного из
следующих символов, вместе с пробелом перед ним – удалить пробел */
      ISXTXT [ I1 ][ I2 ] == '=' ||
      ISXTXT [ I1 ][ I2 ] == '(' ||
      ISXTXT [ I1 ][ I2 ] == ')' ||
      ISXTXT [ I1 ][ I2 ] == '*' ||
      ISXTXT [ I1 ][ I2 ] == '!' || /* добавленный символ операции
AND */
      ISXTXT [ I1 ][ I2 ] == '&') && PREDSYM == ' ') /*
добавленный символ операции AND */
{
    I3-- ;
    goto L1;
}
....
}
```

3. **Синтаксический анализатор:** добавлены функции семантического вычисления нетерминалов LIB, LOP, LVI, STR для первого и второго прохода.

- Для первого прохода тела этих функций пусты и не нуждаются в пояснении.
- На втором проходе тела функций LIB, LOP и STR также пусты.
- При написании тела функции LVI2 за основу брался исходный код функции AVI2, т.к. эти функции выполняют сходные задачи. Ниже будут приведены фрагменты исходного кода функции LVI2 с пояснениями.

```
int LVI2 () {
....
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 == 'T' ) { /* в случае признака типа BIT */
                if (strcmp (SYM[i].RAZR, "16") <= 0) { /* проверка
переменной на разрядность */
                    memcpy (ASS_CARD._BUFCARD.OPERAC, "LH", 2); /*
формирование операции с полусловом */
                } else {
                    memcpy ( ASS_CARD._BUFCARD.OPERAC, "L", 1 ); /*
формирование операции записи переменной в регистр */
                }
                strcpy ( ASS_CARD._BUFCARD.OPERAND, "R1," ); /*
использование регистра R1 */
                strcat ( ASS_CARD._BUFCARD.OPERAND, FORMT [0]);
                ASS_CARD._BUFCARD.OPERAND [strlen
                (ASS_CARD._BUFCARD.OPERAND)] = ' ';
                memcpy ( ASS_CARD._BUFCARD.COMM, "Зарплата переменного В
перуТр", 30 );

                ZKARD ();
                return 0;
            }
            else {
```

```

return 3;
    }
    }
else {
    ....
}

```

В данном фрагменте описывается поведение в том случае, если правая часть выражения – однотермовая. Происходит проверка на тип выражения и формируется ассемблерная команда загрузки значения переменной в регистр R1.

Также данная функция (LVI2) была обновлена для способности обрабатывать 2х-термовую операцию с битовыми переменными:

```

int LVI2() {
    ....
    for ( i = 0; i < ISYM; i++ ) {
        if ( !strcmp (SYM[i].NAME, FORMT[IFORMT-1]) &&
            strlen (SYM[i].NAME) == strlen ( FORMT [IFORMT-1] ))
        {
            if ( SYM [i].TYPE == 'T' ) { /* в случае признака типа BIT */
                if (strcmp (SYM[i].RAZR, "16") <= 0) { /* проверка переменной на
разрядность */
                    memcpy (ASS_CARD._BUFCARD.OPERAC, "LH", 2);
                } else {
                    memcpy ( ASS_CARD._BUFCARD.OPERAC, "L", 1 );
                }
                strcpy ( ASS_CARD._BUFCARD.OPERAND, "R2," ); /* использование
регистра R1 */
                strcat ( ASS_CARD._BUFCARD.OPERAND, FORMT [IFORMT-1]);
                ASS_CARD._BUFCARD.OPERAND [strlen (ASS_CARD._BUFCARD.OPERAND)] = '
';
                memcpy ( ASS_CARD._BUFCARD.COMM, "3ару3ка неРеМеHНou' B перучTp",
30 );
                ZKARD ();

                if ( STROKA [ DST [I2].DST4 - strlen( FORMT [IFORMT-1] ) ] == '!' )
                {
                    memcpy (ASS_CARD._BUFCARD.OPERAC, "O", 1);
                } else {
                    if ( STROKA [DST[I2].DST4 - strlen (FORMT[IFORMT-1]) ] ==
'&' )
                    {
                        memcpy (ASS_CARD._BUFCARD.OPERAC, "N", 1);
                    } else {
                        return 5;
                    }
                }
                strcpy ( ASS_CARD._BUFCARD.OPERAND, "R1," ); /* загрузка данных в
структуру карты */
                strcat ( ASS_CARD._BUFCARD.OPERAND, "R2" );
                ASS_CARD._BUFCARD.OPERAND[strlen(ASS_CARD._BUFCARD.OPERAND)] = ' ';
                memcpy ( ASS_CARD._BUFCARD.COMM, "qpopMupoBaHue нpoMe}l{yTo4Horo
3Ha4eHu9", 39 );
                ZKARD ();
            }
        }
        return 0;
    }
}

```



```
.....  
}
```

В данном фрагменте описывается поведение в том случае, если правая часть выражения – двухтермовая. В этом случае выполняется проверка на тип выражения и формируется ассемблерная команда загрузки значения переменной в регистр R2, а затем идет распознавание команды. В случае, если команда – «логическое И» или «логическое ИЛИ», идет формирование соответствующих ассемблерных команд. Если же команда не распознана, то происходит выход и возвращается код ошибки 5.

Как видно, для логической операции заранее установлен тип 'T', это делается в обновленном функции ODC1, обрабатывающей нетерминал ODC на первом проходе анализатора:

```
int ODC1() {  
    if (!strcmp (FORMT[2], "BIN")) { /* в случае использование  
идентификатора типа BIN */  
        strcpy (SYM [ISYM].NAME, FORMT [1] );  
        strcpy (SYM [ISYM].RAZR, FORMT [4] );  
    } else if (!strcmp (FORMT[2], "BIT")) { /* в случае использование  
идентификатора типа BIT */  
        strcpy (SYM [ISYM].NAME, FORMT[1] );  
        strcpy (SYM [ISYM].RAZR, FORMT[3] );  
    }  
    ...  
    ...  
    else if (!strcmp (FORMT[2], "BIT")) {  
        SYM[ISYM].TYPE = 'T';  
        if (!strcmp (FORMT[4], "INIT")) { /* в случае инициализированной  
переменной */  
            strcpy (SYM[ISYM++].INIT, FORMT[5]); /* формирование  
значения инициализации *.  
        } else {  
            strcpy (SYM[ISYM++].INIT, "'0'B"); /* default значение  
переменной BIT = '0'B */  
        }  
        return 0;  
    }  
    .....  
}
```

Функция OEN2, обрабатывающая нетерминал OEN на первом проходе была обновлена для дополнительной декларации ассемблеровских команд при использовании переменных типа битовых строк:

```
int OEN1() {  
    if (SYM[i].TYPE == 'T') { /* в случае использование идентификатора типа  
BIT */  
        if (i == 1) {  
            memcpy ( ASS_CARD._BUFCARD.OPERAC, "DS", 2 ); /*  
формирование машинной операции */  
            strcpy (ASS_CARD._BUFCARD.OPERAND, "0H");
```

```

        ASS_CARD._BUFCARD.OPERAND [strlen
(ASS_CARD._BUFCARD.OPERAND)] = ' ';
        memcpy (ASS_CARD._BUFCARD.COMM, "BblpaBHuaHue Ha 2
6au'Ta", 25);
        ZKARD ();
    }
    strcpy (ASS_CARD._BUFCARD.METKA, SYM[i].NAME);
    ASS_CARD._BUFCARD.METKA [strlen (ASS_CARD._BUFCARD.METKA)] = ' ';
    if (strcmp (SYM[i].INIT, "'0'B")) { /* в случае инициализированной
переменной */
        memcpy (ASS_CARD._BUFCARD.OPERAC, "DC", 2); /* формирование
машинной операции */
        strcpy (ASS_CARD._BUFCARD.OPERAND, "BL2");
        strncpy (RAB, SYM[i].INIT, strlen (SYM[i].INIT) - 1);
        RAB[strlen (SYM[i].INIT)-1] = '\0';
        strcat (ASS_CARD._BUFCARD.OPERAND, RAB);
        ASS_CARD._BUFCARD.OPERAND [strlen (ASS_CARD._BUFCARD.OPERAND)] =
' ';
        memcpy (ASS_CARD._BUFCARD.COMM, "onpegeJIeHue nepeMeHHou", 24);
        ZKARD ();
    } else {
        memcpy ( ASS_CARD._BUFCARD.OPERAC, "DS", 2 );
        strcpy (ASS_CARD._BUFCARD.OPERAND, "BL2");
        ASS_CARD._BUFCARD.OPERAND [strlen (ASS_CARD._BUFCARD.OPERAND)] =
' ';
        memcpy (ASS_CARD._BUFCARD.COMM, "onpegeJIeHue nepeMeHHou", 24);
        ZKARD ();
    }
    .....
}

```

В функцию обработки нетерминала OPA2 (обработка записи значения арифметического выражения в переменную был добавлен блок сохранения значения логического выражения из регистра в переменную:

```

int OPA2() {
    if (SYM[i].TYPE == 'T') { /* в случае использование идентификатора типа
BIT */
        memcpy ( ASS_CARD._BUFCARD.OPERAC, "STH", 3); /* формирование
машинной операции */
        strcpy ( ASS_CARD._BUFCARD.OPERAND, "R1,");
        strcat ( ASS_CARD._BUFCARD.OPERAND, FORMT[0]) ;
        ASS_CARD._BUFCARD.OPERAND [strlen (ASS_CARD._BUFCARD.OPERAND)] =
' ';
        memcpy ( ASS_CARD._BUFCARD.COMM, "фopмupoBaHue 3Ha4eHu9 apифM
BblpaжeHu9", 42 );
        ZKARD ();
    }
    .....
}

```

В функции gen_COD () добавлен вызов функций семантического вычисления новых нетерминалов LIB, LOP, LVI, STR для первого и второго проходов.

1.5 Результат работы обновленном макета компилятора с ЯВУ с вариантом из рабочего примера.

EX02	START	0	Начало программы
BALR	RBASE	,0	Загрузить регистр базы
USING	*	,RBASE	Назначить регистр базы
LH	R1	,A	Загрузка переменной в реги
LH	R2	,B	Загрузка переменной в реги
OR	R1	,R2	Формирование промежуточног
LH	R2	,C	Загрузка переменной в реги
NR	R1	,R2	Формирование промежуточног
STH	R1	,D	Формирование значения
BCR	15	,14	Выход из пр
DS	0H		Выравнивание
A	DC	BL2 '101 '	Определение
B	DC	BL2 '1010 '	Определение
C	DC	BL2 '111 '	Определение
D	DS	BL2	Определение
RBASE	EQU	2	
R1	EQU	3	
R2	EQU	4	
END	EX02		Конец пр

1.6 Выводы

В ходе выполнения курсовой работы был модифицирован компилятор с языка высокого уровня для работы с заданным примером. С его помощью, после обработки исходного кода программы-задания, описанного в части “Постановка задачи”, был получен ассемблерный код, соответствующий приведенной в описании задания таблице. Таким образом, результаты выполнения данной части курсовой работы можно признать положительными.

Была модифицирована структура базы данных компилятора, заполнены необходимые таблицы. В код компилятора были внесены необходимые изменения.

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

КУРСОВАЯ РАБОТА

**Разработка учебной системы программирования.
Компилятор с языка Ассемблер**
по дисциплине «Системы программирования»

Выполнили
студенты гр.53504/10

Руководитель
доцент



Е.А.Белокопытова
Д.А.Морозов

В.Я. Расторгуев

«___» _____ 2014 г.

Санкт-Петербург
2014

2. Компилятор с языка ассемблер

2.1 Описание примера из личного задания

Данная часть курсового проекта имеет своей целью получение практических навыков построения компилятора с языка АССЕМБЛЕР. Язык АССЕМБЛЕРА сформирован из языковых конструкций АССЕМБЛЕРА IBM 370, а ассемблеровский эквивалент исходной программы формируется компилятором с ЯВУ в виде текстового файла технологической ЭВМ с расширением *.ass.

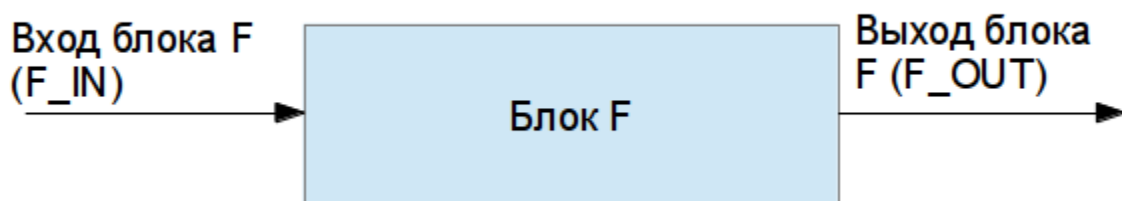
В данной части курсового проекта рассматривается реализация компилятора с ассемблера для исходной программы, полученной в результате выполнения первой части курсового проекта (компилятор с ЯВУ):

Вариант №2

```
EX02: PROC  OPTIONS  ( MAIN );  
        DCL  A      BIT  ( 16 )  INIT  ( '101'B );  
        DCL  B      BIT  ( 16 )  INIT  ( '1010'B );  
        DCL  C      BIT  ( 16 )  INIT  ( '111'B );  
        DCL  D      BIT  ( 16 );  
        D = ( A ! B ) & C;  
END  EX02;
```

2.2 Постановка задачи

В результате выполнения данной части курсового проекта с помощью разработанного компилятора с ассемблера (блок F на схеме), должен быть получен объектный эквивалент исходной программы, который готовится в формате объектных файлов операционной системы ОС IBM 370 ЭВМ и хранится в виде двоичного файла технологической ЭВМ с расширением *.tex.



Вход (F_IN): исходный код на языке Ассемблер:

Метка	КОП	Операнды	Комментарии
EX02	START	0	Начало программы
	BALR	RBASE, 0	Загрузить регистр базы
	USING	*, RBASE	Назначить регистр базой
	LH	R1, A	Загрузка переменной в регистр
	LH	R2, B	Загрузка переменной в регистр
	OR	R1, R2	Формирование промежуточного значения
	LH	R2, C	Загрузка переменной в регистр
	NR	R1, R2	Формирование промежуточного значения
	STH	R1, D	Формирование значения арифметического выражения
	BCR	15, 14	Выход из программы
	DS	0H	Выравнивание на 2 байта
A	DC	BL2'101'	Определение переменной
B	DC	BL2'1010'	Определение переменной
C	DC	BL2'111'	Определение переменной
D	DS	BL2	Определение переменной
RBASE	EQU	2	
R1	EQU	3	
R2	EQU	4	
	END	EX02	Конец программы

Выход (F_OUT): объектный код

task2.tex										0x00000000									
00000000	02	45	53	44	40	40	40	40	40	40	00	10	40	40	00	01			
00000010		45	58	30	32	40	40	40	40	00	00	00	00	40	00	00	22		
00000020		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000030		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000040		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40		
00000050		02	54	58	54	40	00	00	00	40	40	00	02	40	40	00	01		
00000060		05	20	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000070		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000080		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000090		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40		
000000A0		02	54	58	54	40	00	00	02	40	40	00	04	40	40	00	01		
000000B0		48	30	20	18	40	40	40	40	40	40	40	40	40	40	40	40		
000000C0		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
000000D0		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
000000E0		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40		
000000F0		02	54	58	54	40	00	00	06	40	40	00	04	40	40	00	01		
00000100		48	40	20	1A	40	40	40	40	40	40	40	40	40	40	40	40		
00000110		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000120		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000130		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40		
00000140		02	54	58	54	40	00	00	0A	40	40	00	02	40	40	00	01		
00000150		16	34	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000160		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000170		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000180		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40		
00000190		02	54	58	54	40	00	00	0C	40	40	00	04	40	40	00	01		
000001A0		48	40	20	1C	40	40	40	40	40	40	40	40	40	40	40	40		
000001B0		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
000001C0		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
000001D0		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40		
000001E0		02	54	58	54	40	00	00	10	40	40	00	02	40	40	00	01		
000001F0		14	34	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000200		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		
00000210		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40		

Рисунок 2.Объектное представление исходного кода

00000290	07 FE 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002A0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002B0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002C0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
000002D0	02 54 58 54	40 00 00 18	40 40 00 02	40 40 00 01
000002E0	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002F0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000300	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000310	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000320	02 54 58 54	40 00 00 1A	40 40 00 02	40 40 00 01
00000330	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000340	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000350	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000360	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000370	02 54 58 54	40 00 00 1C	40 40 00 02	40 40 00 01
00000380	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000390	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003A0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003B0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
000003C0	02 54 58 54	40 00 00 1E	40 40 00 02	40 40 00 01
000003D0	E0 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003E0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003F0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000400	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000410	02 54 58 54	40 00 00 20	40 40 00 02	40 40 00 01
00000420	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000430	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000440	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000450	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000460	02 45 4E 44	40 40 40 40	40 40 40 40	40 40 40 40
00000470	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000480	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000490	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000004A0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40

Рисунок 3.Объектное представление исходного кода

Ограничения выполнения задания:

- Выравнивание хранения битовых данных в памяти – левое.
- Максимальная длина обрабатываемой переменной типа BIT – 16 бит – 2 байта

С преподавателем согласованы свойства целевой ЭВМ IBM 370:

- Запись полуслова в регистр происходит следующим образом: в регистр записывается полуслово в правую часть.
- Запись данного из регистра в память происходит следующим образом: правая часть слова из регистра записывается в память.

2.3 Перечень и пояснения модификаций состава и структуры компилятора с языка Ассемблер

2.3.1 Добавление строк в таблицу машинных операций

В таблицу машинных операций добавлены 4 новые команды:

- LH — загрузка полуслова в регистр из памяти;
- STH — выгрузка полуслова из регистра в память;
- OR — ИЛИ (оба операнда — в регистрах);
- NR — И (оба операнда — в регистрах).

После добавления таблица машинных операций стала выглядеть так:

```
struct TMOP
{
    unsigned char MNCOP [5];
    unsigned char CODOP    ;
    unsigned char DLOP     ;
    int (*BXPROG) ()       ;
} T_MOP [NOP] =
{
    .....
    {{ 'L', 'H', ' ', ' ', ' ', ' ' } , '\x48' , 4 , FRX} ,
    {{ 'O', 'R', ' ', ' ', ' ', ' ' } , '\x16' , 2 , FRR} ,
    {{ 'N', 'R', ' ', ' ', ' ', ' ' } , '\x14' , 2 , FRR} ,
    {{ 'S', 'T', 'H', ' ', ' ', ' ' } , '\x40' , 4 , FRX} ,
    .....
};
```

2.3.2 Изменение блока подпрограмм, используемых при 1-ом просмотре

В функцию FDC добавлен блок, отвечающий за объявление битовой строки длиной 2 байта. Основное отличие — счетчик адреса увеличивается не на 4, а на 2:

```
int FDC() {
    .....
    if (!strncmp (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "BL2", 3)) { /* в
случае переменной типа BIT */
        T_SYM[ITSYM].DLSYM = 2; /* длина команды - 2 байта */
        T_SYM[ITSYM].PRPER = 'R';
        if ( CHADR % 2 )
        {
            CHADR = (CHADR / 2 + 1) * 2; /* формирование счетчика адреса
согласно типу команды */
            T_SYM[ITSYM].ZNSYM = CHADR;
        }
    }
}
```

```

    }
    PRNMET = 'N';
    CHADR += 2;
    return 0;
}
.....
}

```

В функцию FDS добавлен блок, отвечающий за выделение памяти для битовой строки длиной 2 байта. Основное отличие — счетчик адреса увеличивается не на 4, а на 2:

```

int FDS() {
.....
    if (!strcmp (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "BL2", 3) ||
        !strcmp (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "0H", 2)) {
        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';
        CHADR += 2;
        return 0;
    } else {
        if (!strcmp (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "0H", 2)) {
            if ( CHADR % 2 ) {
                CHADR = (CHADR /2 + 1) * 2;
            }
            CHADR += 2;
        } else {
            if ( CHADR % 4 )
                CHADR = (CHADR /4 + 1) * 4;
            CHADR = CHADR + 4;
        }
    }
.....
}

```

2.3.2 Изменение блока подпрограмм, используемых при 2-ом просмотре

В функцию SDC добавлен блок, отвечающий за объявление битовой строки длиной 2 байта. Для этого случая изменен алгоритм формирования значения переменной. Происходит посимвольное чтение строки со значением из текущей карты и побитовое формирование результата в беззнаковом 2-байтовом целом, представляющем битовую строку. Получившееся значение заносится в соответствующее поле структуры RR; затем происходит вызов функции STXT(2), формирующей новую объектную карту.

```
int SDC() {
.....
    if (!strcmp (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "BL2", 3)) {
        /* в случае переменной типа BIT */
        RAB=strtok ((char*)TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND+4, "'");
        char tmp_str[16];
        int i = 0;
        for (i = 0; i < 16; i++) {
            tmp_str[i] = 0x00;
        }
        i = 0;
        while (*RAB) {
            tmp_str[i++] = *RAB++;
        }
        ushort tmp_val = 0;
        for (i = 0; i < 16; i++) {
            if (tmp_str[i] == '1') {
                tmp_val |= (0x01 << i); /* формирование битового значения
в памяти */
            }
        }
        RR.OP_RR.OP = 0;
        RR.OP_RR.R1R2 = tmp_val;
        STXT(2); /* формирование объектной карты */
        return 0;
    }
.....
}
```

Соответственно, функция SDS переписана следующим образом:

```
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 (!strcmp (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "BL2", 3) ||
               !strcmp (TEK_ISX_KARTA.STRUCT_BUFCARD.OPERAND, "0H", 2))
    {
        RR.OP_RR.OP = 0;
        RR.OP_RR.R1R2 = 0;
        STXT(2);
    } else {
        return 1;
    }
    return 0;
}
.....
}

```

В зависимости от того, под какие данные выделяется память, формируется соответствующий тип объектной карты.

Добавлены указатели на подпрограммы обработки новых команд ассемблера, добавленных в п.2.3.1, при втором просмотре.

```

.....
T_MOP[6].BXPROG = SRX;
T_MOP[7].BXPROG = SRR;
T_MOP[8].BXPROG = SRR;
T_MOP[9].BXPROG = SRX;
.....

```

2.4 Результат работы обновленном макета компилятора с ЯВУ с вариантом из рабочего примера.

В результате работы был получен объектный файл. Для удобного просмотра его содержимого используется утилита ms. Данная программа позволяет просматривать бинарные файлы в шестнадцатиричном виде. Ниже будут приведены скриншоты окна данной программы, содержащие шестнадцатиричное представление полученного объектного файла.

Каждая объектная карта занимает 80 байт или 5 строчек на скриншоте.

task2.tex					0x00000000												
00000000	02	45	53	44	40	40	40	40	40	40	00	10	40	40	00	01	
00000010		45	58	30	32	40	40	40	40	00	00	00	00	40	00	00	22
00000020		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000030		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000040		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40
00000050		02	54	58	54	40	00	00	00	40	40	00	02	40	40	00	01
00000060		05	20	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000070		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000080		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000090		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40
000000A0		02	54	58	54	40	00	00	02	40	40	00	04	40	40	00	01
000000B0		48	30	20	18	40	40	40	40	40	40	40	40	40	40	40	40
000000C0		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
000000D0		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
000000E0		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40
000000F0		02	54	58	54	40	00	00	06	40	40	00	04	40	40	00	01
00000100		48	40	20	1A	40	40	40	40	40	40	40	40	40	40	40	40
00000110		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000120		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000130		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40
00000140		02	54	58	54	40	00	00	0A	40	40	00	02	40	40	00	01
00000150		16	34	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000160		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000170		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000180		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40
00000190		02	54	58	54	40	00	00	0C	40	40	00	04	40	40	00	01
000001A0		48	40	20	1C	40	40	40	40	40	40	40	40	40	40	40	40
000001B0		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
000001C0		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
000001D0		40	40	40	40	40	40	40	40	45	58	30	32	40	40	40	40
000001E0		02	54	58	54	40	00	00	10	40	40	00	02	40	40	00	01
000001F0		14	34	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000200		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
00000210		40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40

Рисунок 4.Объектное представление исходного кода

00000290	07 FE 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002A0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002B0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002C0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
000002D0	02 54 58 54	40 00 00 18	40 40 00 02	40 40 00 01
000002E0	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002F0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000300	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000310	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000320	02 54 58 54	40 00 00 1A	40 40 00 02	40 40 00 01
00000330	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000340	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000350	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000360	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000370	02 54 58 54	40 00 00 1C	40 40 00 02	40 40 00 01
00000380	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000390	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003A0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003B0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
000003C0	02 54 58 54	40 00 00 1E	40 40 00 02	40 40 00 01
000003D0	E0 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003E0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003F0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000400	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000410	02 54 58 54	40 00 00 20	40 40 00 02	40 40 00 01
00000420	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000430	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000440	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000450	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000460	02 45 4E 44	40 40 40 40	40 40 40 40	40 40 40 40
00000470	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000480	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000490	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000004A0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40

Рисунок 5.Объектное представление исходного кода

Выводы

В ходе данной части курсовой работы были внесены изменения в макет программы компилятора с языка Ассемблер, измененная программа реализует новую функциональность согласно заданию. Полученный компилятор после изменения обрабатывает новые команды, формируя на выходе объектный код. Поскольку в результате доработок компилятора с ассемблера был получен необходимый объектный код, то выполнение работы стоит признать успешным.

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

КУРСОВАЯ РАБОТА

**Разработка учебной системы программирования.
Абсолютный загрузчик и отладчик**
по дисциплине «Системы программирования»

Выполнили
студенты гр.53504/10

Руководитель
доцент




Е.А.Белокопытова
Д.А.Морозов



В.Я. Расторгуев

«___» _____ 2014 г.

Санкт-Петербург
2014

3. Абсолютный загрузчик

3.1 Описание примера из личного задания

Данная часть курсового проекта имеет своей целью получение практических навыков построения:

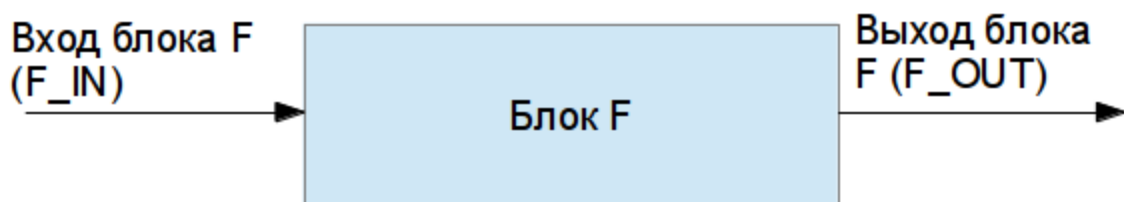
- загрузчика (утилиты, формирующей исполняемый эквивалент исходной программы),
- эмулятора (программной модели объектного или, иначе, целевого процессора, предназначенного для исполнения на нем исполняемого эквивалента исходной программы),
- отладчика исполняемого эквивалента исходной программы, запускаемого на эмуляторе.

Вариант №2

```
EX02: PROC  OPTIONS  ( MAIN );  
    DCL  A      BIT   ( 16 )  INIT  ( '101'B );  
    DCL  B      BIT   ( 16 )  INIT  ( '1010'B );  
    DCL  C      BIT   ( 16 )  INIT  ( '111'B );  
    DCL  D      BIT   ( 16 );  
    D = ( A ! B ) & C;  
END EX02;
```

3.2 Постановка задачи

Необходимо выполнить доработку элементов макета учебной системы программирования (блок F на схеме), в части абсолютного загрузчика и эмулятора до уровня, позволяющего обрабатывать “новые” для макета конструкции объектного кода, примененные в соответствующем варианте, получая на выходе результат работы программы макета в эмуляторе.



Вход (F_IN): объектное представление исходного кода

task2.tex				0x00000000			
00000000	02 45 53 44	40 40 40 40	40 40 00 10	40 40 00 01			
00000010	45 58 30 32	40 40 40 40	00 00 00 00	40 00 00 22			
00000020	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000030	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000040	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40			
00000050	02 54 58 54	40 00 00 00	40 40 00 02	40 40 00 01			
00000060	05 20 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000070	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000080	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000090	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40			
000000A0	02 54 58 54	40 00 00 02	40 40 00 04	40 40 00 01			
000000B0	48 30 20 18	40 40 40 40	40 40 40 40	40 40 40 40			
000000C0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
000000D0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
000000E0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40			
000000F0	02 54 58 54	40 00 00 06	40 40 00 04	40 40 00 01			
00000100	48 40 20 1A	40 40 40 40	40 40 40 40	40 40 40 40			
00000110	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000120	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000130	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40			
00000140	02 54 58 54	40 00 00 0A	40 40 00 02	40 40 00 01			
00000150	16 34 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000160	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000170	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000180	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40			
00000190	02 54 58 54	40 00 00 0C	40 40 00 04	40 40 00 01			
000001A0	48 40 20 1C	40 40 40 40	40 40 40 40	40 40 40 40			
000001B0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
000001C0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
000001D0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40			
000001E0	02 54 58 54	40 00 00 10	40 40 00 02	40 40 00 01			
000001F0	14 34 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000200	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			
00000210	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40			

Рисунок 6.Объектное представление исходного кода

Выход (F_OUT): работа эмулятор в соответствии с заданным поведением

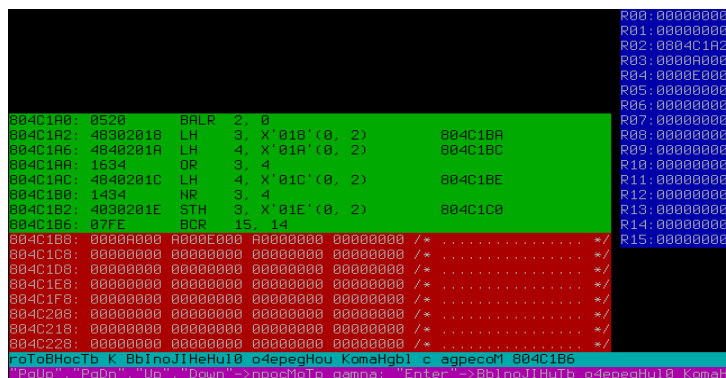


Рисунок 8. Программа-макет эмулятора и отладчика

Рассчитаем ожидаемый результат работы программы:

- Переменная А представляется в памяти в следующем виде: '1010000000000000' (двухбайтовая битовая строка)
- Переменная В представляется в памяти в следующем виде: '1010000000000000' (двухбайтовая битовая строка)

00000290	07 FE 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002A0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002B0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002C0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
000002D0	02 54 58 54	40 00 00 18	40 40 00 02	40 40 00 01
000002E0	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000002F0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000300	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000310	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000320	02 54 58 54	40 00 00 1A	40 40 00 02	40 40 00 01
00000330	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000340	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000350	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000360	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000370	02 54 58 54	40 00 00 1C	40 40 00 02	40 40 00 01
00000380	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000390	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003A0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003B0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
000003C0	02 54 58 54	40 00 00 1E	40 40 00 02	40 40 00 01
000003D0	E0 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003E0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000003F0	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000400	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000410	02 54 58 54	40 00 00 20	40 40 00 02	40 40 00 01
00000420	00 00 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000430	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000440	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000450	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40
00000460	02 45 4E 44	40 40 40 40	40 40 40 40	40 40 40 40
00000470	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000480	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
00000490	40 40 40 40	40 40 40 40	40 40 40 40	40 40 40 40
000004A0	40 40 40 40	40 40 40 40	45 58 30 32	40 40 40 40

Рисунок 7. Объектное представление исходного кода

- Переменная С представляется в памяти в следующем виде: '1110000000000000' (двухбайтовая битовая строка)

- Тогда в результате операции побитового ИЛИ над A и B ($A \mid B$) получится следующее значение: '1010000000000000'.
- В результате операции побитового И над получившимся значением и C : $((A \mid B) \& C)$ получится следующее значение: '1010000000000000'.

3.3 Изменения, внесенные в программу-макет эмулятора системы IBM 370

В ходе выполнения курсовой работы в исходный код программ-макета были внесены следующие изменения:

- В таблицу машинных операций добавлены 4 новые команды:
 - LH — загрузка полуслова в регистр из памяти;
 - STH — выгрузка полуслова из регистра в память;
 - OR — ИЛИ (оба операнда — в регистрах);
 - NR — И (оба операнда — в регистрах).

```
struct TMOP
{
    unsigned char MNCOP [5];
    unsigned char CODOP    ;
    unsigned char DLOP     ;
    int (*BXPROG) ()       ;
} T_MOP [NOP] =

{
    .....
    {{ 'L' , 'H' , ' ' , ' ' , ' ' } , '\x48', 4 , FRX},
    {{ 'O' , 'R' , ' ' , ' ' , ' ' } , '\x16', 2 , FRR},
    {{ 'N' , 'R' , ' ' , ' ' , ' ' } , '\x14', 2 , FRR},
    {{ 'S' , 'T' , 'H' , ' ' , ' ' } , '\x40', 4 , FRX},
    .....
};
```

- Добавлены функции P_LH(), P_STH(), P_OR(), P_NR(), вызываемые при обработке соответствующих команд:
 - Функция P_LH() обеспечивает загрузку полуслова (2 байта) в регистр.

```
int P_LH () {
```

```

int sm;
ADDR = VR[B] + VR[X] + D; /* формирование адреса *.
sm = (int) ( ADDR - I );
VR[R1] = OBLZ[BAS_IND + CUR_IND + sm] * 0x100 +
OBLZ[BAS_IND + CUR_IND + sm+1]; /* формирование регистра */
return 0;
}

```

- Функция P_STH() обеспечивает выгрузку полуслова из регистра в память.

```

int P_STH () {
    int sm,i;
    char bytes[2];
    ADDR = VR[B] + VR[X] + D; /* формирование нового адреса */
    sm = (int) (ADDR -I);
    bytes[0] = ((VR[R1] % 0x10000L)-(VR[R1]%0x10000L)%0x100))/0x100; /*
формирование первого байта памяти */
    bytes[1] = (VR[R1] % 0x10000L) % 0x100; /* формирование второго байта
памяти */
    for (i=0; i<2; i++)
        OBLZ[BAS_IND + CUR_IND + sm + i] = bytes[i];
    return 0;
}

```

- Функция P_OR() реализует побитовое ИЛИ над содержимым двух регистров. Результат записывается в первый регистр.

```

int P_OR () {
    unsigned short r1 = (unsigned short)VR[R1];
    unsigned short r2 = (unsigned short)VR[R2];
    r1 = r1 | r2; /* выполнение операции OR */
    VR[R1] = r1; /* запись результата в регистр */
    return 0;
}

```

- Функция P_NR() реализует побитовое И над содержимым двух регистров. Результат записывается в первый регистр.

```

int P_NR () {
    unsigned short r1 = (unsigned short)VR[R1];
    unsigned short r2 = (unsigned short)VR[R2];
    r1 = r1 & r2; /* выполнение операции AND */
    VR[R1] = r1; /* запись результата в регистр */
    return 0;
}

```

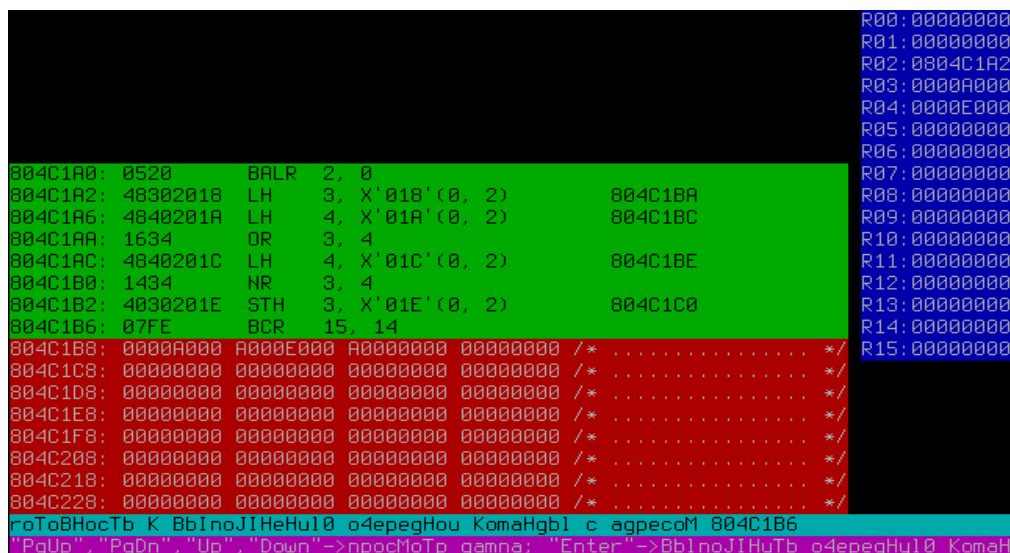
- В часть функции sys(), отмеченную меткой SKIP добавлена обработка новых

ассемблерных команд:

```
int sys(void) {  
.....  
    switch (T_MOP[k].CODOP) {  
  
.....  
        case '\x40' : P_STH();  
        break;  
        case '\x48' : P_LH();  
        break;  
        case '\x16' : P_OR();  
        break;  
        case '\x14' : P_NR();  
        break;  
    }  
.....  
}
```

3.3 Результаты

По результатам работы над модификацией эмулятора исполняемый код задания (объектный код) получил возможность выполняться на макете:



Address	Instruction	Comment
804C1A0	BALR 2, 0	
804C1A2	LH 3, X'018'(0, 2)	804C1BA
804C1A6	LH 4, X'01A'(0, 2)	804C1BC
804C1AA	OR 3, 4	
804C1AC	LH 4, X'01C'(0, 2)	804C1BE
804C1B0	NR 3, 4	
804C1B2	STH 3, X'01E'(0, 2)	804C1C0
804C1B6	BCR 15, 14	
804C1B8	00000000 00000000 00000000 00000000	/* */
804C1C8	00000000 00000000 00000000 00000000	/* */
804C1D8	00000000 00000000 00000000 00000000	/* */
804C1E8	00000000 00000000 00000000 00000000	/* */
804C1F8	00000000 00000000 00000000 00000000	/* */
804C208	00000000 00000000 00000000 00000000	/* */
804C218	00000000 00000000 00000000 00000000	/* */
804C228	00000000 00000000 00000000 00000000	/* */

Registers (R00-R15):

Register	Value
R00	00000000
R01	00000000
R02	00000000
R03	00000000
R04	00000000
R05	00000000
R06	00000000
R07	00000000
R08	00000000
R09	00000000
R10	00000000
R11	00000000
R12	00000000
R13	00000000
R14	00000000
R15	00000000

Command Line: roToBHoCTb K BbInoJIHeHy10 o4epeghou Komahgb1 c agpecom 804C1B6
"PgUp", "PgDn", "Up", "Down" -> npecMOTp gamma; "Enter" -> BbInoJIHeTy o4epeghy10 Komah

Рисунок 9. Программа-макет эмулятора и отладчика

Выводы

В ходе выполнения данной части курсового проекта был модифицирован исходный код программы-макета, части загрузчика и эмулятора. Результаты, полученные после его запуска полностью совпадают с ожидаемыми – был получен заданный ход выполнения программы, описанный в разделе “Постановка задачи”.

Таким образом, результаты выполнения данного этапа курсовой работы можно признать положительными.

Заключение

Измененная программа-макет реализует новую функциональность согласно заданию. Макет учебной системы программирования после изменения обрабатывает новые конструкции языка высокого уровня и на выходе выдает результат работы программы в эмуляторе.

Учебная система программирования протестирована после изменения. Тестовые примеры содержали как смешанные определения переменных обоих типов с инициализацией и без, так и определения только одного из типов с инициализацией и без. На этапе тестирования ошибок в программе «Учебная система программирования» не выявлено.

Результата выполнения работы следует признать успешным.

Список литературы

1. Расторгуев В.Я. Учебное пособие по курсу «Системы программирования». Разработка элементов учебной системы программирования. Компилятор с ЯВУ. Методический указания. СПб. 42С
2. Расторгуев В.Я. Учебное пособие по курсу «Системы программирования». Разработка элементов учебной системы программирования. Компилятор с с языка Ассемблер. Методический указания. СПб. 42С
3. Расторгуев В.Я. Учебное пособие по курсу «Системы программирования». Разработка элементов учебной системы программирования. Абсолютный загрузчик. Методический указания. СПб. 42с