Санкт-Петербургский политехнический университет Петра Великого

Институт информационных технологий и управления

Кафедра «Информационные и управляющие системы»

**Курсовая работа**

**Разработка учебной системы программирования**

**Компилятор языка высокого уровня.**

по дисциплине «Системы программирования»

Вариант 9

Выполнили

студенты гр. 53504/3

Мамонтов Я. С.

Кузнецов Д. А.

Хутар Давуд Захи

Руководитель:

Расторгуев В. Я.

**«\_\_\_» \_\_\_\_\_\_\_\_\_\_** 2016 г.

Санкт-Петербург

2016

# Введение

Данная курсовая работа имеет своей целью получение практических навыков построения компилятора с языка высокого уровня (ЯВУ), являющегося одним из элементов системы программирования, образующих в совокупности технологический конвейер (см. рисунок 1).



Рис . Структура курсового проекта

При этом предполагается то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (ЕС ЭВМ). В этой системе:

* в качестве языка высокого уровня (ЯВУ) выбран язык, образованный из подмножества языковых конструкций ПЛ1, а исходная программа готовится в виде текстового файла технологической ЭВМ с расширением \*.pli;
* язык АССЕМБЛЕРА сформирован из языковых конструкций АССЕМБЛЕРА ЕС ЭВМ, а ассемблеровский эквивалент исходной программы формируется в виде текстового файла технологической ЭВМ с расширением \*.ass;
* объектный эквивалент исходной программы готовится в формате объектных файлов операционной системы ОС ЕС ЭВМ и хранится в виде двоичного файла технологической ЭВМ с расширением \*.tex;
* загрузочный эквивалент исходной программы представляет собой машинный код ЕС ЭВМ, запоминаемый в области ОЗУ технологической ЭВМ, являющейся зоной загрузки для эмулятора объектной ЭВМ.

# Постановка задачи

В данной работе требуется доработать существующие элементы учебной системы программирования для обработки новых конструкций языка высокого уровня. В этой части работы будет рассматриваться преобразование исходного текста на языке высокого уровня в эквивалент исходного текста на Ассемблере (см. рисунок 2).

Исходный текст

на

ЯВУ

Эквивалент исходного текста на Ассемблере

Объектный эквивалент

исходного текста

Загрузочный эквивалент исходного текста

Кросс-компилятор с ЯВУ

Кросс-компилятор с Ассемблера

Загрузчик

Эмулятор

Отладчик

Рис 2. Рассматриваемая часть учебной системы

В качестве исходного текста на ЯВУ на вход кросс-компилятора подается текст программы, написанный на языке PL/1:

EX09: PROC OPTIONS ( MAIN );

DCL A BIT ( 5 ) INIT ( ‘101’B );

DCL B DEC FIXED;

B = A;

END EX09;

В результате работы компилятора должен получится эквивалент программы на языке Assembler архитектуры IBM 370:

EX09 START 0

BALR @RBASE, 0

USING \*, @RBASE

L @RRAB, A

SRL @RRAB, 29

CVD @RRAB, @BUF

LA @RADD, @BUF

MVC B(3), 5(@RADD)

BCR 15, 14

A DC BL4'101'

B DC PL3'0'

DS 0F

@BUF DC PL8'0'

@RBASE EQU 15

@RRAB EQU 5

@RADD EQU 4

END

# Анализ поставленной задачи

Основной задачей являлось преобразование переменной из битовой строки (BIT) в десятичное число (DEC). Для этого нужно осуществить цепочку преобразований BIT->BIN->DEC на языке Ассемблер.

После проведенного анализа было получено следующее эквивалентное преобразование:

Язык PL/I

B = A;

Язык Ассемблера

LH @RRAB, A - Загрузка значения переменной в регистр

SRL @RRAB, 13 - Арифметический сдвиг вправо

CVD @RRAB, @BUF - Смена типа

LA @RADD, @BUF - Загрузка адреса BUF в регистр RADD

MVC B(3), 5(@RADD) - Перемещение

# Входные ограничения

1. Объявление переменной как битовой строки согласно спецификации языка должно быть представлено в виде BL<кол-во байтов>.<кол-во битов>. Для решения нашей задачи мы в целях упрощения подменили объявление на BL<кол-во битов>.
2. Десятичные числа только положительные
3. Размер памяти, выделяемый по умолчанию под хранение десятичного числа: 3 байта. Данное ограничение было введено вследствие того, что в задании при объявлении десятичного числа не указывается размерность, следовательно, при разработке компилятора необходимо это учесть, и точно задать размер выделяемой памяти по умолчанию.
4. Проверка и обработка ситуаций выхода размера чисел за отведенный предел не производится.
5. Предполагается использовать для служебных целей метки, начинающиеся с символа «@». Таким образом пользователь не может использовать символ @ в начале идентификаторов.

# Преобразование грамматики

В грамматику языка были внесены следующие изменения (выделены жирным):

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>

8. <ODC> ::= DCL\_<IPE>\_BIN\_FIXED(<RZR>);|

DCL\_<IPE>\_BIN\_FIXED(<RZR>)INIT(<LIT>) ;**|**

**DCL\_<IPE>\_BIT(<RZR>)INIT(<LIT>);|**

**DCL\_<IPE>\_DEC\_FIXED;**

9. <IPE> ::= <IDE>

10. <RZR> ::= <CIF> | <RZR><CIF>

11. <LIT> ::= <MAN>B

12. <MAN> ::= 1 | <MAN>0 | <MAN>1

13. <OPA> ::= <IPE>=<AVI>;

14. <AVI> ::= <LIT> | <IPE> | <AVI><ZNK><LIT> |

<AVI><ZNK><IPE>

15. <ZNK> ::= + | -

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";

"<AVI>" - нетерминал "арифметическое выражение";

"<ZNK>" - нетерминал "знак";

"<OEN>" - нетерминал "оператор эпилога программы".

Далее рассмотрим часть продукций, которые были изменены в ходе работы:

D-><BUK>  
DCL\_<IPE>\_BIN\_FIXED(<RZR>);-> <ODC>

DCL\_<IPE>\_BIN\_FIXED(<RZR>)INIT(<LIT>); **->** <ODC>

**DCL\_<IPE>\_BIT(<RZR>)INIT(<LIT>); -> <ODC>**

**DCL\_<IPE>\_DEC\_FIXED; -> <ODC>**

Сгруппируем получившиеся продукции в «кусты»:

D-><BUK>

|

->CL\_<IPE>\_->**DEC\_FIXED;-><ODC>**

|

->**BI->T(<RZR>)INIT(<LIT>);-><ODC>**

|

->N\_FIXED(<RZR>)->;-><ODC>

|

->INIT(<LIT>);**->**<ODC>

# Модификация базы данных исходного макета

Нами были добавлены дополнительные определения операций в синтаксис. Полученные ранее «кусты» были представлены в табличном виде и записаны в таблицу синтаксических правил:

struct

{

int POSL;

int PRED;

char DER[4];

int ALT;

} SINT[NSINT] =

/\* \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_

| NN : посл : пред : дер : альт |

|\_\_\_\_\_\_\_\_\_\_:\_\_\_\_\_\_\_\_\_:\_\_\_\_\_\_\_:\_\_\_\_\_\_\_:\_\_\_\_\_\_| \*/

{

...

{/\*. 201 .\*/ 202 , 51 , "T " , 0 },

{/\*. 202 .\*/ 203 , 201 , "( " , 0 },

{/\*. 203 .\*/ 204 , 202 , "RZR" , 0 },

{/\*. 204 .\*/ 205 , 203 , ") " , 0 },

{/\*. 205 .\*/ 206 , 204 , "; " , 65 },

{/\*. 206 .\*/ 207 , 205 , "ODC" , 0 },

{/\*. 207 .\*/ 0 , 206 , "\* " , 0 },

{/\*. 208 .\*/ 209 , 49 , "D " , 0 },

{/\*. 209 .\*/ 210 , 208 , "E " , 0 },

{/\*. 210 .\*/ 211 , 209 , "C " , 0 },

{/\*. 211 .\*/ 212 , 210 , " " , 0 },

{/\*. 212 .\*/ 213 , 211 , "F " , 0 },

{/\*. 213 .\*/ 214 , 212 , "I " , 0 },

{/\*. 214 .\*/ 215 , 213 , "X " , 0 },

{/\*. 215 .\*/ 216 , 214 , "E " , 0 },

{/\*. 216 .\*/ 217 , 215 , "D " , 0 },

{/\*. 217 .\*/ 218 , 216 , "; " , 65 },

{/\*. 218 .\*/ 219 , 217 , "ODC" , 0 },

{/\*. 219 .\*/ 0 , 218 , "\* " , 0 }

};

Остальные таблицы базы данных компилятора менять не пришлось. Модифицировать таблицу входов в «кусты» не понадобилось, т.к. при добавлении новых синтаксических правил, новых входов в «кусты» не появилось. Таблица матрицы смежности также осталась без изменений, т.к. добавленные новые правила не привнесли новых символов в алфавит.

# Модификация алгоритма исходного макета

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

int ODC1();

int AVI2();

int OPA2();

int OEN2();

Также для повышения функциональности и улучшения внешнего вида команды создания временно переменной BUFF были вынесены в отдельную функцию AddTemp8BytesVariable.

Измененные отрывки функций представлены ниже:

**/\* Функция создания вспомогательной 8 байтовой переменной для CVD \*/**

void **AddTemp8BytesVariable**( char\* name )

{

strcpy ( SYM [ISYM].NAME, name ); //Пишем имя, например TEMPDEC

strcpy ( SYM [ISYM].RAZR, "8" ); //Разрядность 8 байт

strcpy ( SYM [ISYM].INIT, "0B" ); //Значение 0

SYM [ISYM].TYPE = 'D'; //Тип DEC

ISYM++; //Увеличиваем количество символов на 1

}

**/\* Программа семантического вычисления нетерминала ODC на первом проходе \*/**

int **ODC1**()

{

...

/\* Случай типа BIT \*/

else if ( !strcmp( FORMT[2], "BIT" ) )

{

SYM[ISYM].TYPE = 'L'; /\* Устанавливаем тип переменной \*/

strcpy( SYM[ISYM].RAZR, FORMT[3] ); /\* Устанавливаем разрядность \*/

if( !strcmp(FORMT[4], "INIT") ) /\* Инициализация \*/

strcpy(SYM[ISYM].INIT, FORMT[5]);

else

strcpy(SYM[ISYM].INIT, "0B");

}

/\* Случай типа DEC FIXED \*/

else if ( !strcmp( FORMT[2], "DEC" ) && !strcmp( FORMT[3], "FIXED" ) )

{

SYM[ISYM].TYPE = 'D'; /\* Устанавливаем тип переменной \*/

strcpy( SYM[ISYM].RAZR, FORMT[3] );

SYM[ISYM].RAZR[0] = '3'; /\* Устанавливаем разрядность \*/

SYM[ISYM].RAZR[1] = '\0';

strcpy(SYM[ISYM].INIT, "0B"); /\* Инициализация \*/

}

...

}

**/\* Программа семантического вычисления нетерминала AVI на втором проходе \*/**

int **AVI2**()

{

...

/\* Случай типа BIT \*/

else if( SYM[i].TYPE == 'L' )

{

/\* Установка операции \*/

memcpy(ASS\_CARD.\_BUFCARD.OPERAC, "LH", 2);

/\* Установка регистра \*/

strcpy( ASS\_CARD.\_BUFCARD.OPERAND, "RRAB," );

/\* Установка адреса ячейки памяти \*/

strcat( ASS\_CARD.\_BUFCARD.OPERAND, FORMT[0]);

/\* вставляем разделитель \*/

ASS\_CARD.\_BUFCARD.OPERAND[strlen(ASS\_CARD.\_BUFCARD.OPERAND)] = ' ';

/\* и построчный коментарий\*/

memcpy( ASS\_CARD.\_BUFCARD.COMM, "Load the variable to the register", 33 );

/\*Запоминаем операцию ассемблера\*/

ZKARD();

return 0;

}

...

}

**/\* Программа семантического вычисления нетерминала OPA на втором проходе \*/**

int **OPA2**()

{

...

/\* Случай типа DEC FIXED \*/

else if( SYM[i].TYPE == 'D' )

{

int j;

for( j = 0; j < ISYM; j++ )

{

if( !strcmp(SYM[j].NAME, FORMT[1]) && strlen(SYM[j].NAME) == strlen(FORMT[1]) )

{

/\* Случай типа BIT \*/

if( SYM[j].TYPE == 'L' )

{

/\* Arithmetic shift \*/

/\* Установка операции \*/

memcpy(ASS\_CARD.\_BUFCARD.OPERAC, "SRL", 3);

/\* Установка регистра \*/

strcpy( ASS\_CARD.\_BUFCARD.OPERAND, "RRAB," );

/\* Вычисление смещения \*/

int int\_offset = 16 - strlen(SYM[j].INIT) + 1;

char ch\_offset[2];

sprintf(ch\_offset, "%d", int\_offset);

strcat( ASS\_CARD.\_BUFCARD.OPERAND, ch\_offset);

/\* вставляем разделитель \*/

ASS\_CARD.\_BUFCARD.OPERAND[strlen(ASS\_CARD.\_BUFCARD.OPERAND)] = ' ';

/\* и построчный коментарий\*/

memcpy( ASS\_CARD.\_BUFCARD.COMM, "Arithmetic shift", 16 );

/\*Запоминаем операцию ассемблера\*/

ZKARD();

/\* Change type \*/

memcpy(ASS\_CARD.\_BUFCARD.OPERAC, "CVD", 3);

strcpy( ASS\_CARD.\_BUFCARD.OPERAND, "RRAB," );

strcat( ASS\_CARD.\_BUFCARD.OPERAND, "BUF");

/\* вставляем разделитель \*/

ASS\_CARD.\_BUFCARD.OPERAND[strlen(ASS\_CARD.\_BUFCARD.OPERAND)] = ' ';

/\* и построчный коментарий\*/

memcpy( ASS\_CARD.\_BUFCARD.COMM, "Change type", 11 );

ZKARD();

/\* Load addres to register \*/

AddTemp8BytesVariable("BUF");

memcpy(ASS\_CARD.\_BUFCARD.OPERAC, "LA", 2);

strcpy( ASS\_CARD.\_BUFCARD.OPERAND, "RADD," );

strcat( ASS\_CARD.\_BUFCARD.OPERAND, "BUF");

/\* вставляем разделитель \*/

ASS\_CARD.\_BUFCARD.OPERAND[strlen(ASS\_CARD.\_BUFCARD.OPERAND)] = ' ';

/\* и построчный коментарий\*/

memcpy( ASS\_CARD.\_BUFCARD.COMM, "Load addres to register", 23 );

ZKARD();

/\* Moving to needed variable \*/

memcpy(ASS\_CARD.\_BUFCARD.OPERAC, "MVC", 3);

strcpy( ASS\_CARD.\_BUFCARD.OPERAND, FORMT[1] );

strcat( ASS\_CARD.\_BUFCARD.OPERAND, "(3),5(RADD)");

/\* вставляем разделитель \*/

ASS\_CARD.\_BUFCARD.OPERAND[strlen(ASS\_CARD.\_BUFCARD.OPERAND)] = ' ';

/\* и построчный коментарий\*/

memcpy( ASS\_CARD.\_BUFCARD.COMM, "Moving to needed variable", 25 );

ZKARD();

return 0;

}

return 3;

}

}

}

...

}

**/\* Программа семантического вычисления нетерминала OEN на втором проходе \*/**

int **OEN2**()

{

...

/\* Случай типа BIT \*/

else if( SYM[i].TYPE == 'L' )

{

/\* Установка имени переменной \*/

strcpy( ASS\_CARD.\_BUFCARD.METKA,SYM[i].NAME );

ASS\_CARD.\_BUFCARD.METKA[strlen(ASS\_CARD.\_BUFCARD.METKA)] = ' ';

/\* Установка операции \*/

memcpy( ASS\_CARD.\_BUFCARD.OPERAC, "DC", 2 );

/\* Установка размера и начального значения \*/

strcpy( ASS\_CARD.\_BUFCARD.OPERAND,"BL" );

strcat( ASS\_CARD.\_BUFCARD.OPERAND, "16" );

strcat( ASS\_CARD.\_BUFCARD.OPERAND, "\'" );

SYM[i].INIT[strlen(SYM[i].INIT) - 1] = '\0';

strcat( ASS\_CARD.\_BUFCARD.OPERAND, SYM[i].INIT );

/\* замыкающий апостроф \*/

ASS\_CARD.\_BUFCARD.OPERAND[strlen(ASS\_CARD.\_BUFCARD.OPERAND)] = '\'';

/\* построчнsq комментарий\*/

memcpy(ASS\_CARD.\_BUFCARD.COMM,"Definition of variable", 22);

ZKARD();

}

/\* Случай типа DEC FIXED \*/

else if( SYM[i].TYPE == 'D' )

{

/\* Добавляем выравнивание \*/

if (!strcmp(SYM [i].NAME, "BUF"))

{

memcpy ( ASS\_CARD.\_BUFCARD.OPERAC, "DS", 2 );

/\* Выравнивание 4 байта \*/

memcpy ( ASS\_CARD.\_BUFCARD.OPERAND, "0F", 2 );

memcpy ( ASS\_CARD.\_BUFCARD.COMM, "Aligment", 8 );

ZKARD ();

}

strcpy( ASS\_CARD.\_BUFCARD.METKA,SYM[i].NAME );

ASS\_CARD.\_BUFCARD.METKA[strlen(ASS\_CARD.\_BUFCARD.METKA)] = ' ';

memcpy( ASS\_CARD.\_BUFCARD.OPERAC, "DC", 2 );

strcpy( ASS\_CARD.\_BUFCARD.OPERAND,"PL" );

strcat( ASS\_CARD.\_BUFCARD.OPERAND, SYM[i].RAZR );

strcat( ASS\_CARD.\_BUFCARD.OPERAND, "\'" );

SYM[i].INIT[strlen(SYM[i].INIT) - 1] = '\0';

strcat( ASS\_CARD.\_BUFCARD.OPERAND, SYM[i].INIT );

ASS\_CARD.\_BUFCARD.OPERAND[strlen(ASS\_CARD.\_BUFCARD.OPERAND)] = '\'';

memcpy(ASS\_CARD.\_BUFCARD.COMM,"Definition of variable", 22);

ZKARD();

}

...

}

# Выводы

В ходе выполнения данной части курсовой работы в базу данных и алгоритм компилятора с PL/1 были внесены необходимые изменения, позволяющие использовать в коде программы языковые конструкции, представленные в задании. Общая архитектура данной части системы позволяет достаточно легко вносить необходимые изменения в алгоритм работы и расширять набор поддерживаемых конструкций ЯВУ. При выполнении работы столкнулись с проблемой использования функции CVD, у которой на выходе 8 байтовая переменная, вместо необходимой 3 байтовой и с проблемой выравнивания переменных в памяти.

Полученный в ходе работы компилятор был успешно скомпилирован. После его запуска с поданным на вход заданным вариантом задания № 9 был получен исходный код на языке Ассемблера, который полностью соответствует разработанному вручную варианту, описанному в главе «Постановка задачи».