О том, как ставить задачу для первой части курсовой работы - разработки компилятора с языка высокого уровня

Схема компиляции входного задания имеет следующий вид:

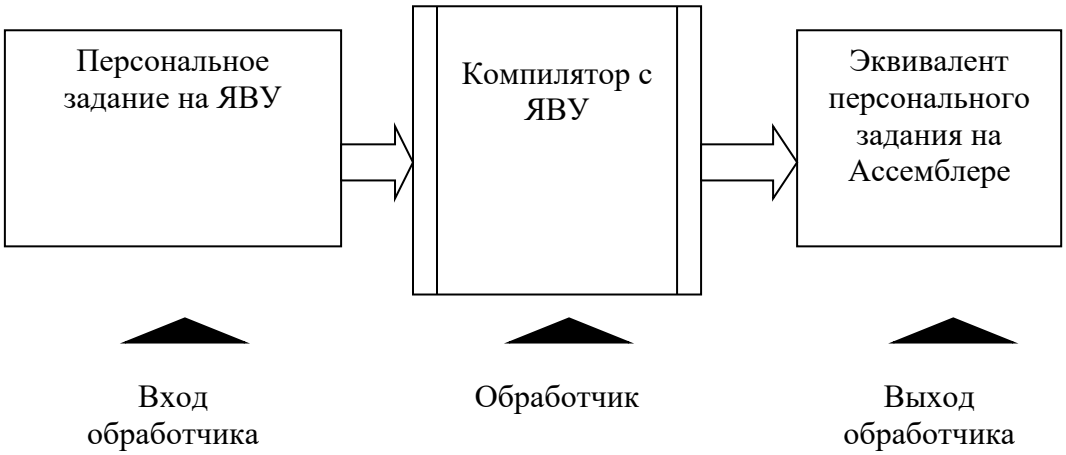

Рис. 1. Схема компиляции задания с ЯВУ на Ассемблер

Каждое персональное задание, выданное студента, оригинально, но схема компиляции одинакова. Поясним ее на демо-примере на который настроен переданный макет-заготовка, переданный студенту вместе с заданием, имеющим вид:

```
ex: proc options (main);
    dcl A bin fixed (31) init ( 11B );
    dcl B bin fixed (31) init ( 100B );
    dcl C bin fixed (31) init ( 101B );
    dcl D bin fixed (31);

    D=A+B-C;
end ex;
```

Рис.2 Текст на входе компилятора с ЯВУ

Итак исходный текст на ЯВУ этого задание поступает на вход компилятора с ЯВУ.

Компилятор с ЯВУ, будучи текстовым обработчиком, в процессе компиляции на выходе должен сформировать ассемблеровский эквивалент

входного задания. Учитывая возможность сделать это многими различными вариантами, студенту нужно выбрать и вручную построить единственный вариант, представляющейся ему оптимальным (время, память и т.д). После построения этот вариант будет представлен в виде следующей табличной записи:

Метка	КОП	Операнды	Пояснения

Рис. 3. Текст на выходе компилятора

Теперь можно сформулировать постановку задачи (или, иначе технического задания) для этой части курсовой работы следующим образом:

ТЗ на разработку компилятора с ЯВУ

Предмет разработки:	**Компилятор с ЯВУ**
Технические требования:	**При подаче на вход компилятора задания (см. рис.2) на выходе должен быть построен эквивалент входного задания на Ассемблере (см. рис. 3)**

Рис.4. Постановка задачи на первую часть курсовой работы

Остается определиться с технологией построения эквивалента входного задания (см. рис. 3). Сделаем это.
Даже не зная ЯВУ, легко увидеть в демо-примере структуру процедурного блока, состоящую из:

- Пролога блока,
- Раздела объявлений объекта,
- Раздела манипулирования объектами,

- Эпилога блока

ex: proc options (main);

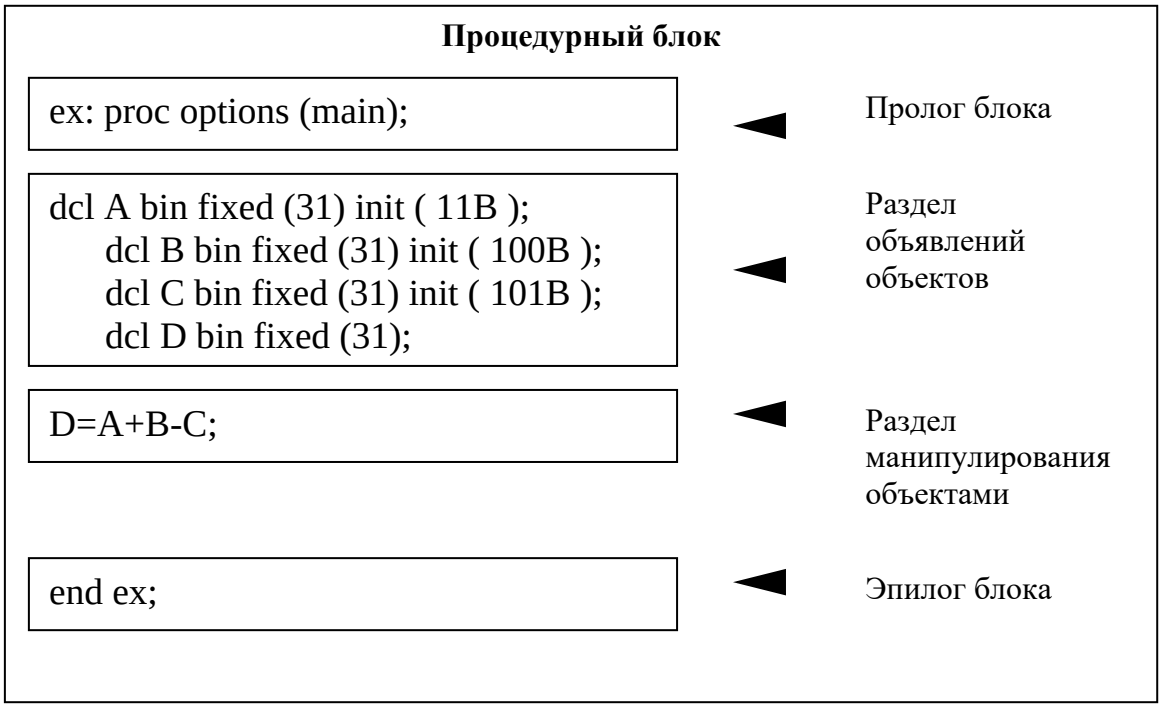

Рис. 5. Блочная структура входного задания

Даже не зная Ассемблера, можно с высокой степенью правоты предполагать, что он, как и ЯВУ, тоже имеет блочную структуру. Поэтому задача преобразования входного текста на ЯВУ в выходной текст на Ассемблере может быть разделена на четыре подзадачи:

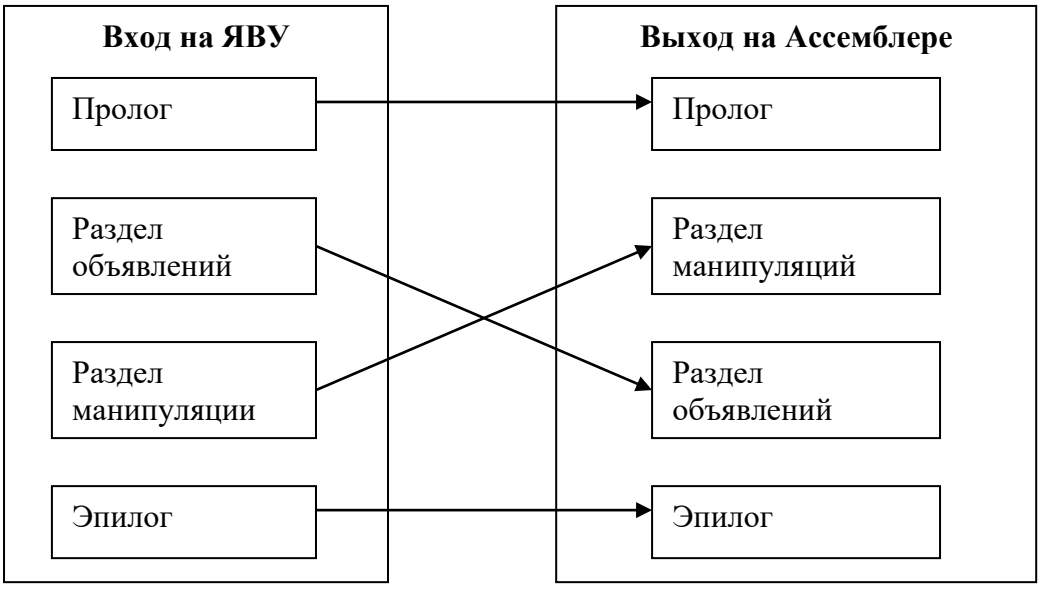

Рис.6. Подзадачи преобразования текста ЯВУ в текс Ассемблера

Решение первой из четырех задач – построение пролога входного задания на Ассемблере состоит во взятие одного из возможных шаблонов блока на Ассемблере:

Метка	КОП	Операнды	Пояснения
???	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RRBASE регистром базы

и занесения вместо «???» имени процедурного блока, которое в демо-примере есть «ex». Получаем в результате низкоуровневый блок:

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы

В моих методических пособиях, переданных в электронный фонд библиотеки СПбПУ, можно уточнить смысл каждой из строк низкоуровневого пролога. Шаблон эпилога одинаков для всех вариантов заданий и имеет вид:

Метка	КОП	Операнды	Пояснения
	END		Конец текста блока

Переходим к разделу объявлений.
Каждую строку высокоуровневого объявления «dcl» (аббревиатура «declare») заменяем низкоуровневым аналогом «dc» (аббревиатура «define constant») в случае объявления объекта с инициализацией или низкоуровневым аналогом «ds» (аббревиатура «define storage») в случае объявления объекта без инициализации. Результат заполнения будет таким:

Метка	КОП	Операнды	Пояснения
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D

Один из возможных вариантов построения раздела манипулирования м.б. таким:

Метка	КОП	Операнды	Пояснения
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D

Соединяем все части низкоуровневого блока на Ассемблере вместе:

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
	END		Конец текста блока

В этом варианте низкоуровневого блока есть две проблемы.
Первая состоит переходе траектории выполнения программы автоматически из конца раздела манипулирования («ST RRAB,A») в начало раздела объявлений («A DC F’3’»), что ведет к тяжелой ошибке, если не переключить траекторию на возврат к вызвавшей программе или к операционной системе.
Добавим эту команду передачи управления «BCR» и получим:

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы

	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
	END		Конец текста блока

Вторая оставшаяся проблема заключена в применение нами регистров в символическом (RBASE, RRAB, RVIX), а не в цифровом виде. Исправим это, применяя псевдооператор Ассемблера «EQU» в разделе объявления объектов:

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
	END		Конец текста блока

Низкоуровневый эквивалент входного задания построен, что завершает работу над техническим заданием к первой части курсовой работы.

Устранение конфликта между основными и вспомогательными метками

В рекомендациях 1 описана методика ручного преобразования высокоуровневой записи демо-примера:

```
ex: proc options (main);
    dcl A bin fixed (31) init ( 11B );
    dcl B bin fixed (31) init ( 100B );
    dcl C bin fixed (31) init ( 101B );
    dcl D bin fixed (31);

    D=A+B-C;
end ex;
```

Рис.1 Текст на входе компилятора с ЯВУ

в низкоуровневый эквивалент на Ассемблере:

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
	END		Конец текста блока

Рис. 2. Низкоуровневый эквивалент демо-примера на Ассенмблере

Правильность и, следовательно, работоспособность низкоуровневого эквивалента, полученного после компиляции (см. рис.2), пока не вызывает сомнений, а вот к области применения компилятора, реализующего такую методику преобразования, м.б. высказаны замечания.

Действительно, сделаем мысленно некий эксперимент, а именно:
Предположим, что по неизвестным нам причинам автор демо-примера при его написании для объекта, названного, скажем именем **A**, выберет альтернативно одно из имен: **BRASE, RRAB или RVIX**. Тогда примененная нами ранее ручная методика построения низкоуровневого эквивалента приведет к двойному объявлению в низкоуровневом эквиваленте имени, выбранного для объекта, в противоречащих друг другу смыслах.
Например, для случая с **RBASE:**

- первое объявление будет таким, как на рис 3,
- второе объявление будет таким, каккак на рис. 4.

Метка	КОП	Операнды	Пояснения
RBASE	DC	F’3’	A=3

Рис.3

Метка	КОП	Операнды	Пояснения
RBASE	EQU	5	RBASE назначим 5

Рис.4.

Причина возникновения самой возможности проявления двойного объявления одного и того же имени заключена в том, что при построении низкоуровневого эквивалента нам потребовалось дополнительно к именам объектов, объявленных автором демо-примера в тексте демо-примера, добавить «служебные» имена вспомогательных «служебных» объектов.

Так как двойное объявление имени – это такая ошибка, которая ведет к невозможности успешной компиляции, то, чтобы продолжать успешно работать с компилятором, генерирующим двойные объявления, нужно запретить прикладным программистам использовать в нашем случае для своих целей имена: **RBASE**, **RRAB**, **RVIX**. Этот путь сокращает комфорт использования компилятора, а вместе с этим и область его применения, точнее, количество прикладных пользователей, которые будут готовы его приобрести и эксплуатировать.

Другой способ устранения возможности двойного объявления имен объектов может быть основан на том, чтобы строить «служебные» имена по правилам, разводящим результат построения прикладников и разработчиков компилятора по разным непересекающимся множествам.
Сделать это можно, например, разрешением разработчикам компилятора использовать при формировании «служебных» имен некоторого спецсимвола, например, «@», который

далее рекомендовать ставить первым в «служебном» имени. Применяя это разрешение к варианту демо-примера на рис.2, получим новый его вариант (см. рис. 5)

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	@RBASE,0	отн. адр. базы в @RBASE
	USING	*,@RBASE	Объявл. @RBASE регистром базы
	L	@RRAB,A	Загрузка A в регистр @RRAB
	A	@RRAB,B	A+B грузим в @RRAB
	S	@RRAB,C	A+B-C грузим в @RRAB
	ST	@RRAB,D	A+B-C из @RRAB грузим в D
	BCR	15,@RVIX	Переход по адр. в рег. @RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
@RBASE	EQU	5	@RBASE назначим 5
@RRAB	EQU	3	@RRAB назначим 3
@RVIX	EQU	14	@RVIX назначим 14
	END		Конец текста блока

Рис. 5. Вариант низкоуровневого эквивалента демо-примера с использованием спецсимвола «@»

Применение разных правил формирования «прикладных» и «служебных» имен обеспечит невозможность двойного объявления одного и того же имени одновременно, в качестве, как «прикладного», так и «служебного». Разрешение использования спецсимвола «@» разработчику компилятора с одновременным запретом такой возможности для прикладного программиста не должен вызывать протестов последнего, т.к. запрет на использование спецсимволов в именах объектов прикладных программ является общепринятым правилом.

Низкоуровневый эквивалент построен и согласован, что дальше?

Чтобы ответить на вопрос, поставленный в заголовке текста, сделаем два действия:

1) Посмотрим на формулировку персонального задания, которая одинакова для всех вариантов и размещена в начале перечня всех вариантов персональных заданий, содержащегося в файле variantn.rtf внутри архива с исходными данными для КР, переданного в группы в начале семестра. Вот она:

«**Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте**».

2) Для того, чтобы полнее раскрыть смысл лаконичной формулировки персонального задания, приведенной выше, вспомним про учебное пособие [1] (далее в тексте «**Пособие**»), доступного в библиотеке СПбПУ и также переданного в группы внутри архива с исходными данными для КР, раскроем его, найдем в оглавлении **Пособия** раздел «**Рекомендации по порядку выполнения работы**» и прочитаем следующее:
« . . .
Данная лабораторная работа предполагает использование в качестве начальной основы отлаженной и работающей программы-макета компилятора с языка ПЛ1, с ограниченным набором языковых конструкций, достаточным для написания законченной правильной программы контрольного примера. Программа-макет написана на алгоритмическом языке Си и снабжена подробным контекстным комментарием.

Студенту предлагается расширить первичный набор языковых конструкций, поддерживаемый макетным компилятором с ЯВУ, до уровня, позволяющего оттранслировать пример, содержащийся в его личном задании, и произвести соответствующую доработку макетного компилятора с ЯВУ. При этом необходимо:

а) Ознакомиться с исходными текстами программы-макета и контрольного примера, на который настроена программа-макет,

б) Проверить работоспособность программы-макета путем проведения компиляции контрольного примера,

в) Визуально выделить из исходного текста примера, содержащегося в личном задании, множества языковых конструкций, отсутствующих в контрольном примере,

г) Спроектировать перечень коррекций элементов БД и подпрограмм программы-макета с целью обеспечения компиляции примера из личного задания,

д) Ввести коррекции в исходный текст программы-макета, снабдив измененные места исходного текста построчным комментарием, и записать модифицированный вариант в свой личный директорий на сервере,

е) Выполнить отладку модифицированной программы-макета,

ж) Составить отчет по результатам работы

. . . »

Переходим к выполнению рекомендаций из пп. от **а)** до **ж).**

Действия по пункту а)

В архиве с исходными данными для КР находим папку:
StudLabRab_2020_magistry\LinuxLabRab\LinuxLabRab.TGZ\LinuxLabRab.tar\.\linux_lab_rab

Убеждаемся, во-первых, в том, что в этой папке имеется С-файл **Komppl.c.** Делая беглый входной контроль содержимого файла **Komppl.c**, убеждается в том, что это и есть исходный текст С-программы макета компилятора с языка высокого уровня PL/1, который должен быть доработан исполнителями КР под свой персональный вариант задания.
Далее находим PL/1-файл **examppl.pli** контрольного или, иначе, демо-примера и знакомимся с ним:

```
EXAMP1: PROC OPTIONS ( MAIN );

      DCL A BIN FIXED ( 31 ) INIT ( 111B );
      DCL B BIN FIXED ( 31 ) INIT ( 100B );
      DCL C BIN FIXED ( 31 ) INIT ( 101B );
      DCL D BIN FIXED ( 31 );

      D = A + B - C;

END EXAMP1;
```

Рис.1

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту б)

Устанавливаем на компьютере ОС Linux для 32-хразрядной IBM PC, например:

- Устанавливаем дистрибутив **ubuntu 14.0**,
- Дополняем пакетом панельного файл-менеджера **Midnight Comander (mc)** с помощью директивы **sudo apt-get install mc**,
- Дополняем пакетом формирования псевдографических окон **ncurses** с помощью директивы **sudo aptr-get install libncurses5-dev**,
- Переписываем туда архив с исходными данными для КР, включая папку **linux_lab_rab** .

Для перекодировки исходных файлов из **KOI-8** в **UNICOD (UTF8)** выполняем shell-скрипт **ChangeCodTable**.
Выполняем ту часть shell-скрипта **GenSysProg** (генерация системы программирования), которая относится к построению из исходного С-файла **Komppl.c** исполняемого файла **Komppl.exe**.
Выполняем ту часть shell-скрипта **StartTestTask**, которая относится к запуску компилятора с ЯВУ (т.е. файла **Komppl.exe)** для компиляции демо-примера (т.е. файла **examppl.pli**).

Убеждаемся в том, что эквивалент демо-примера на Ассемблере **examppl.ass** построен компилятором успешно и что он содержит следующий текст на Ассемблере:

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F'3'	A=3
B	DC	F'4'	B=4
C	DC	F'5'	C=5
D	DS	F	Резерв памяти для D
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
	END		Конец текста блока

Рис.2

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту в)

Продолжение настоящих рекомендаций основано на специально построенном для этой цели условном варианте персонального задания, приведенном ниже и имеющем в своей основе демо-пример, приведенный выше (см. рис. 1):

Условный персональный вариант

```
EXAMP1: PROC OPTIONS ( MAIN );
      DCL A BIN FIXED ( 31 ) INIT ( 111B );
      DCL B BIN FIXED ( 31 ) INIT ( 100B );
      DCL C BIN FIXED ( 31 ) INIT ( 101B );
      DCL D BIN FIXED ( 31 );
      DCL E BIN FIXED ( 31 );
      D = A + B - C;
      E = D * A / B;
END EXAMP1;
```

Рис.3

При визуальном сравнении исходных текстов условного персонального задания (см. рис. 3) и демо-примера (см. рис. 1) отмечаем то, что отличия в них минимальны и состоят в расширении:

- терминального алфавита грамматики ЯВУ PL/1 путем введения в него новых терминальных символов «*****» и «/»,
- перечня допустимых арифметических операций, применяемых в арифметическом выражении, находящемся в правой части оператора присваивания, операцией **умножения «*»** и операцией **деления «/»**.

Понятно, что отличия, обнаруженные в исходных текстах условного персонального задания и демо-примера, обусловят отличия их эквивалентов на языке Ассемблер. На рис. 4 приведен один из возможных вариантов такого эквивалента для условного персонального задания, приведенного на рис.3.

,

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 4

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту г)

В основу функционирования макета компилятора с ЯВУ положен (см. **Пособие**) алгоритм, настраиваемый на грамматику ЯВУ, которая выступает в качестве параметра настройки. Грамматика имеет три группы параметров, определяющих такие свойства ЯВУ, как:

- Лексика,
- Синтаксис,
- Семантика.

Детальное пояснение можно найти в **Пособие**, а в данных кратких рекомендациях для пояснений воспользуемся следующим рисунком (см. стр.13 **Пособия)** :

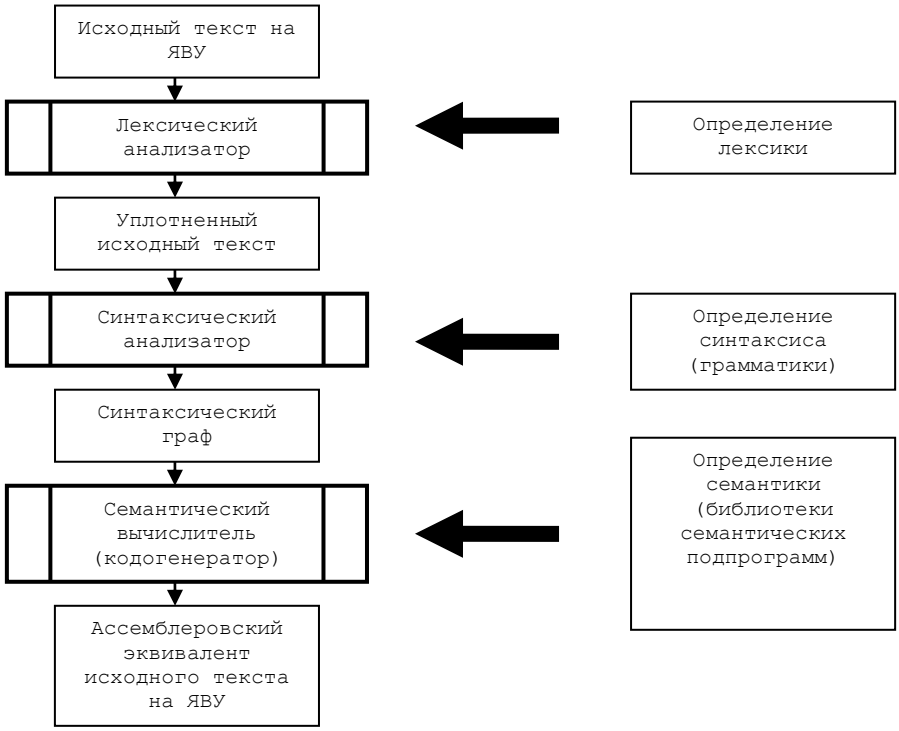

Рис.5. Уточненная функциональная схема компилятора с ЯВУ

Макет компилятора ЯВУ, переданный в группы в архивном файле, уже настроен соответствующими параметрами на грамматику демо-примера в части лексических, синтаксических и семантических аспектов. Все эти параметры размещены, как в элементах БД (лексика синтаксис), так и в множестве семантических функций/подпрограмм (семантика) компилятора с ЯВУ.
Для перенастройки на условный персональный вариант КР необходима коррекция существующих настроек компилятора ЯВУ путем изменения некоторых параметров. Возникает вопрос: «Каких именно?». Проведем экспресс-анализ результатов поиска отличий демо-примера и условного персонального варианта, полученных выше (при выполнении **действий по пункту в**).

Там нами были отмечены отличия условного персонального задания от демо-примера в части расширений:

- терминального алфавита (это лексический аспект ЯВУ),
- перечня допустимых арифметических операций (это синтаксический аспект).

Ответить на вопрос о необходимости коррекции семантических параметров или отсутствии таковой можно будет только после проведения предстоящих коррекций синтаксиса. При этом, критерием необходимости семантических правок будет введение в синтаксис дополнительных нетерминальных символов.

Невозможно, да и методически неправильно в рамках кратких настоящих рекомендаций разбирать детали довольно большой программы макета компилятора ЯВУ с целью указания мест расположения параметров для внесения в них коррекций. Это входит в работу студентов. Поэтому, говоря о коррекции конкретного параметра, будем обсуждать смысловой аспект коррекции и указывать на места описания особенностей корректируемого параметра в **Пособие.** Однако при этом заметим то, что, если единственным местом хранения исходного текста программы макета компилятора ЯВУ является файл **Komppl.c**, то внутри него содержатся и значения всех настроечных параметров.

Лексические правки

Дополнительные терминальные символы «*****» и «/» являются разделителями термов (см. стр. 12 и стр. 13 **Пособия**) и должны быть включены в существующее множество допустимых разделителей:

```
{ ":" , "+" , "-" , ";" , "(" , ")" , "_" }
```

Модифицированное множество допустимых разделителей будет таким:

```
{ ":" , "+" , "-" , ";" , "(" , ")" , "_" , "*" , "/"}
```

Синтаксические правки

Синтаксис демо-примера (см. стр. 7 и стр.8 **Пособия**) имеет следующее определение (см. рис. 6):

```
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>);

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".

Рис. 6

Для реализации условного варианта задания необходимо в синтаксисе ЯВУ разрешить использование в арифметических выражениях дополнительных операций **умножения** «*****» и **деления** «/». Это разрешение м.б. выполнено небольшим изменением правой части синтаксического правила № 15. При этом, новый вариант синтаксиса м.б. описан так, как на рис 7, на котором дополнительные правки выделены жирным шрифтом.

```
 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>);
 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>
```

Рис. 7

Отметим важную деталь, состоящую в том, что для нашего случая нам не пришлось вводить в синтаксис новые нетерминальные символы, а это, в свою очередь, не потребует изменять состав множества семантических функций (семантических подпрограмм). Достаточно будет изменить алгоритм только в существующих семантических подпрограммах, соответствующих существующим нетерминальным символам.

Синтаксические правки
Вносим коррекции, учитывающие обработку операций умножения и деления в существующие семантические подпрограммы **ZNK** и/или **AVI**. Состав семантических подпрограмм не меняется, ввиду отсутствия необходимости.

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту д)

Методика предварительной подготовки и ввода коррекций в исходный текст программы-макета **Komppl.c** описана в **Пособие.**

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту е)

Отладка программы макета компилятора ЯВУ, модифицированного правками, производится студентами с опорой на собственный опыт написания и отладки Linux-приложений.
Отладку можно считать законченной в случае, если в процессе экспериментальной проверки работоспособности после запуска отлаживаемой программы компилятора ЯВУ на выходе генерируется ассемлеровский эквивалент входного персонального задания, который совпадает с эталонным вариантом, согласованным с преподавателем на этапе постановки задачи (в ТЗ).
Работа отлаженной версии компилятора предъявляется (демонстрируется) преподавателю с одновременным вручением отчета для проверки/принятия.

В крайнем случае, если появились проблемы и/или вопросы, то идем с ними за как в консультацией к преподавателю.

Действия по пункту ж)

Отчет по результатам составляется с учетом «**Требований к содержанию отчета**», приведенных, как в разделе 4 **Пособия**, так и ниже:

« . . .
Отчет должен соответствовать рекомендациям по оформлению студенческих курсовых работ, размещенных на сайте университета, и в общем случае должен содержать следующие разделы:

- *описания примера из личного задания (указать формулировку личного задания),*
- *постановки задачи модификации программы-макета (описание выбранного способа модификации программы-макета),*

- *содержащие перечень и пояснения модификаций состава и (или) структуры БД компилятора,*
- *перечень и пояснения модификаций алгоритма программы-макета компилятора (возможные формы описаний в виде: блок-схемы, текста на Си с комментариями, а в очевидных случаях обычным текстом),*
- *выводы по результатам работы (указать позитивные и негативные моменты, проявившиеся в процессе работы, с общей личной оценкой проделанной работы).*

. . . »

Литература

1. Расторгуев В.Я. Разработка элементов учебной системы программирования. Компилятор с языка высокого уровня [электронный ресурс]: учебное пособие, СПбПУ, Спб., 2012

Практический результат по КР получен. Как его оформить?

Результаты выполненной КР должны быть отражены в отчете. Как его делать?

Общий титульный лист

Курсовая работа (КР) должна быть начата студентами после получения задания на КР, где определяется тема. Для дисциплины «Системы программирования» задание на КР для всех студентов формулируется одинаково следующим образом:

«**Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте**».

Различия эта общая формулировка получает в момент привязки к персональному варианту.

Формулировка задания на КР, приведенная выше, говорит о том, что **целью или, иначе, темой КР является создание модифицированной учебной системы программирования**.

Итак, на данный момент нам известны все параметры, необходимые для оформления титульного листа отчета по КР, точнее:

тип работы: Курсовая работа,
дисциплина : Системы программирования,
тема : Разработка учебной системы программирования.

Используя эти параметры, оформляем общий титульный лист (см. Приложение 1).

Состав и структура отчета

Для обеспечения возможности защиты и сдачи отчета по частям по мере реализации элементов учебной системы программирования логично привязать состав и структуру отчета к составу и структуре предмета разработки - учебной системы программирования, которые приведены на рис. 1. В этом случае отчет по КР можно создавать, защищать и сдавать по частям.
Из рис. 1 легко видеть, что последовательность реализации элементов учебной системы программирования хронологически логично построить в следующем порядке:

1) Компилятор с ЯВУ.
2) Компилятор с Ассемблера.
3) Модуль загрузчика, эмулятора, отладчика.

Следовательно, в такой же последовательности нужно выполнять и оформление результатов перечисленных этапов КР в виде трех следующих отчетов:

1) Отчет по компилятору с ЯВУ.
2) Отчет по компилятору с Ассемблера.
3) Отчет по модулю загрузчика, эмулятора, отладчика.

Таким образом, с учетом общего титульного листа и всех только что перечисленных отчетов по отдельным элементам учебной системы программирования структура сводного отчета будет такой, как на рис. 2.

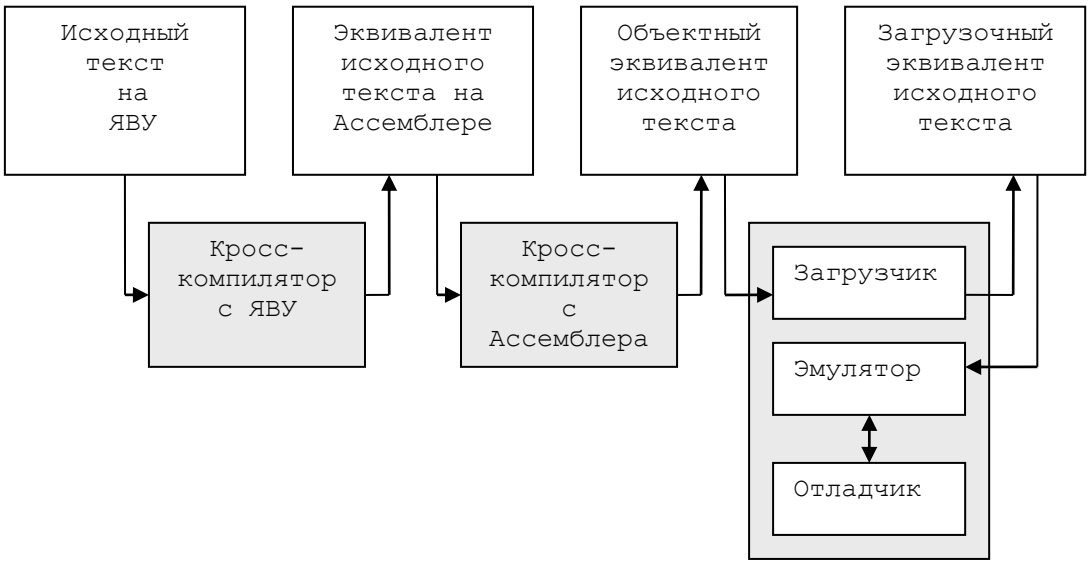

Рис.1. Состав учебной системы программирования

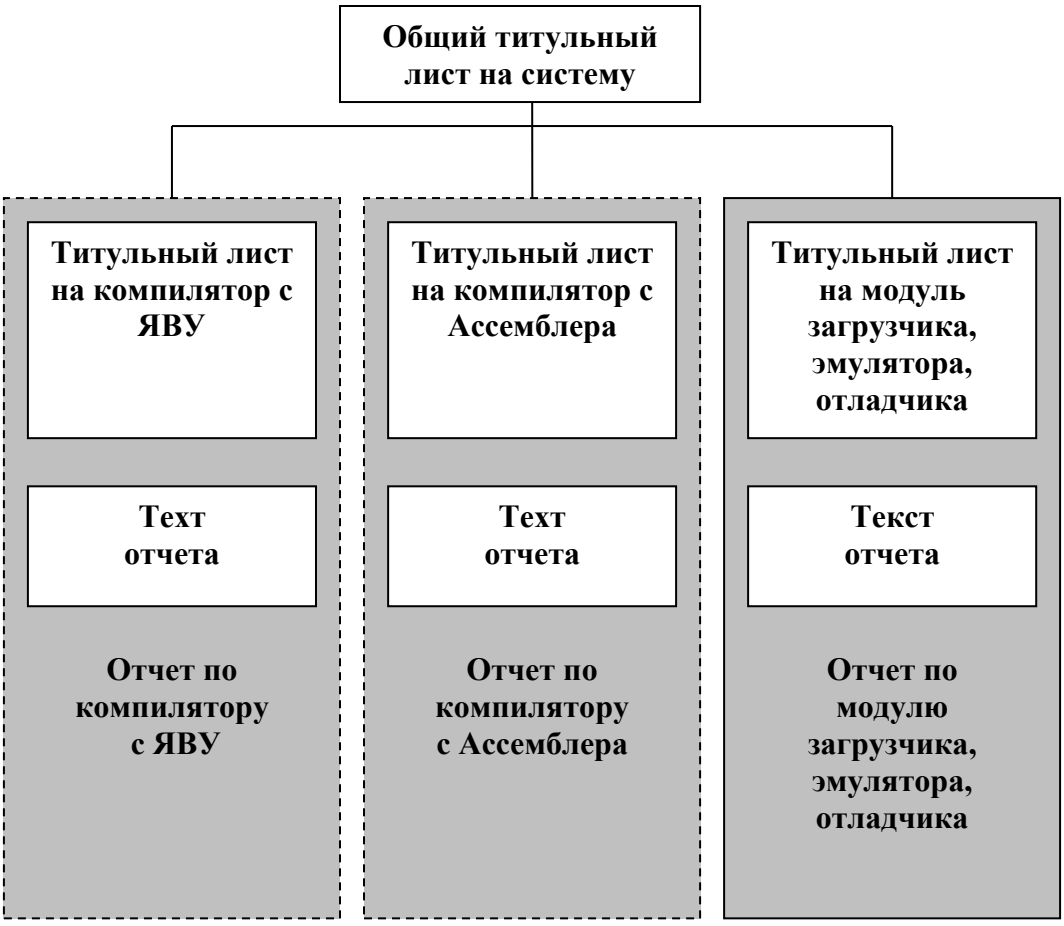

Рис. 2.

Титульный лист на элемент системы

Содержание титульного листа отчета для любой из трех частей КР подобны, поэтому, в приложение 2 приводится пример только для одной части, относящейся к компилятору с ЯВУ. Любой из оставшихся частных титульных листов будет аналогичен приведенному в приложении 2.

Перечень разделов отчета

Согласно разделу 4 из методического пособия [1], в материал отчета следует включить:
« . . .

1) Описания примера из личного задания (указать формулировку личного задания),
2) Постановку задачи модификации программы-макета (описание выбранного способа модификации программы-макета),
3) Перечень и пояснения модификаций состава и (или) структуры БД компилятора,
4) Перечень и пояснения модификаций алгоритма программы-макета компилятора (возможные формы описаний в виде: блок-схемы, текста на Си с комментариями, а в очевидных случаях обычным текстом),
5) Выводы по результатам работы (указать позитивные и негативные моменты, проявившиеся в процессе работы, с общей личной оценкой проделанной работы).

. . . »

Эти рекомендации из Пособия [1] были выпущены в 2012 г. Практика показала необходимость включения в перечень требований еще одной позиции «Результаты экспериментальной проверки модифицированного компилятора с ЯВУ».

Добавим эту позицию в перечень требований к отчет и получим:

1) Описания примера из личного задания (указать формулировку личного задания),
2) Постановки задачи модификации программы-макета (описание выбранного способа модификации программы-макета),
3) Перечень и пояснения модификаций состава и (или) структуры БД компилятора,
4) Перечень и пояснения модификаций алгоритма программы-макета компилятора (возможные формы описаний в виде: блок-схемы, текста на Си с комментариями, а в очевидных случаях обычным текстом),
5) Результаты экспериментальной проверки модифицированного компилятора с ЯВУ
6) Выводы по результатам работы (указать позитивные и негативные моменты, проявившиеся в процессе работы, с общей личной оценкой проделанной работы).

Далее будут приведены примеры выполнения требований к разделам отчета, основанные на условном персональном варианте, взятом из Рекомендаций_3:

Условный персональный вариант

```
EXAMP1: PROC OPTIONS ( MAIN );
        DCL A BIN FIXED ( 31 ) INIT ( 111B );
        DCL B BIN FIXED ( 31 ) INIT ( 100B );
        DCL C BIN FIXED ( 31 ) INIT ( 101B );
        DCL D BIN FIXED ( 31 );
        DCL E BIN FIXED ( 31 );
        D = A + B - C;
        E = D * A / B;
END EXAMP1;
```

Пояснения к разделу «Задание на КР»

В этом разделе отчета логично дать формулировку всего задания на КР, уточнить свой персональный вариант и далее сфокусировать внимание на тех его аспектах, результаты работы над которыми войдут в настоящий отчет.
Пример реализации раздела отчета приведен в приложении 3.

Пояснения к разделу «Постановка задачи модификации макета в части компилятора с ЯВУ»

В этом разделе следует привести постановку задачи (техническое задание) на ту часть КР, которая документируется данным отчетом. В данном случае это компилятор с ЯВУ, который должен быть получен путем модификации функционала макетного образца компилятора с ЯВУ под персональный вариант задания.
Понятно, что, согласно существующей практике, черновик ТЗ составляет исполнитель (студент), после чего согласует его с преподавателем, далее приступает к его реализации и переносит его в отчет. В ТЗ должны быть определены:

- предмет разработки,
- требования по входу и выходу,
- опционно, при согласовании могут быть предложены Заказчику (преподавателю) и согласованы с ним мотивированные ограничения функционала предмета разработки.

Пример реализации раздела отчета приведен в приложении 4.

Пояснения к разделу «Модификация состава и структуры БД макета компилятора с ЯВУ»

Согласно заданию на КР, студент должен получить компилятор с ЯВУ с заданным функционалом, работая «не с чистого листа», а методом модификации БД и алгоритма готового макета.
БД представляет собой совокупность множеств терминальных символов (лексика) и грамматических таблиц (синтаксис), приведенных в [1].
Технологически данные, предназначенные для заполнения этих таблиц, синтезируются сначала в высокоуровневом формате (метаязык Бэкуса/Науры), который ручными технологиями преобразуются в низкоуровневый машинный формат, который далее заносятся в декларативную часть С-программы компилятора. Ручные технологии продемонстрированы в [1] на демо-примере.
В данном разделе отчета следует привести:

- Высокоуровневые определения для таблиц БД грамматики,
- Низкоуровневый эквивалент высокоуровневых определений для таблиц БД,
- Промежуточные результаты, полученные во время преобразований.

Высокоуровневые определения модифицированной грамматики для своего персонального варианта следует строить, путем модификации высокоуровневого определения грамматики демо-примера, которую можно взять в [1].

Пример реализации раздела отчета приведен в приложении 5.

Пояснения к разделу «Модификация алгоритма макета компилятора с ЯВУ

Область адаптации алгоритма макета компилятора с ЯВУ ограничивается множеством семантических функций, т.к. количество семантических функций равно количеству нетерминальных символов, использованных в определениях синтаксиса языка. Отсюда вытекает относительно простой способ определения тех семантических функций, которые следует включить в рабочий список, элементы которого нужно, либо синтезировать, либо модифицировать, а именно:

1) Семантическую функцию следует модифицировать, если изменения в синтаксисе коснулись правой части определения нетерминального символа, соответствующего этой семантической функции.
2) Семантическую функцию следует синтезировать, если изменения в синтаксисе потребовали добавить новый, т.е. ранее отсутствующий в синтаксисе нетерминальный символ, который соответствует семантической функции.
3) Семантическую функцию, **возможно,** потребуемся модифицировать, если правая часть нетерминального символа, соответствующего этой семантической функции, содержит нетерминальные символы для пп. 1) или 2).

Поясним этот принцип на примере персонального варианта. Изучая модифицированный синтаксис (см. приложение 5), видим:

- во-первых то, что состав нетерминальных символов после модификации синтаксиса не изменился, а это значит, что новых нетерминальных символов нет. Вывод – задача синтеза семантических функций отсутствует,
- во-вторых, то, что правая часть определения нетерминального символа <ZNK> модифицирована. Вывод – стоит задача модификации, либо подпрограммы ZNK1 (первый просмотр), либо подпрограммы ZNK2 (второй просмотр), либо подпрограммы AVI1(первый просмотр), либо подпрограммы AVI2 (второй просмотр).

Убеждаемся прямой проверкой в том, что из четырех выделенных подпрограмм: ZNK1, ZNK2, AVI1, AVI2 три полпрограммы, а именно: ZNK1, ZNK2 и AVI1 имеют «пустую» семантику. Следовательно, **возможна** модификация только подпрограммы AVI2.

Пример документирования модификации подпрограммы AVI2 в отчете приведен в приложении 6.

Пояснения к разделу «Результаты экспериментальной проверки модифицированного компилятора с ЯВУ»

После отладки модифицированного макета до состояния, когда на выходе модифицированного компилятора с ЯВУ получится эквивалент персонального варианта на Ассемблере, пишем этот раздел отчета.

Пример реализации этого раздела приведен в приложении 7.

Пояснения к разделу «Выводы по результатам работы»

В начале раздела необходимо разместить две обязательные фразы.

Первая фраза отражает самооценку разработчика в одном из двух вариантов:

1) Порученное задание мною выполнено успешно.
2) Порученное задание мною не выполнено.

Вторая фраза доказательно подтверждает первую фразу в одном из двух возможных вариантов:

1) Успешность выполнения порученного задания подтверждена полным совпадением выходного текста, полученного при экспериментальной проверке компилятора с ЯВУ, с эталоном на рис. 4.
2) Неудача выполнения порученного задания подтверждена несовпадением выходного текста, полученного при экспериментальной проверке компилятора с ЯВУ, с эталоном на рис. 4.

Третья и последующие фразы добавляются исполнителем опционно и без особой надежды на то, что эти фразы будут внимательно прочитаны Заказчиком.

Пример реализации этого раздела приведен в приложении 8.

Пояснения ко всему отчету в целом

Пример целого отчета, в котором собраны все формулировки разделов, рекомендованные выше, приведен в приложении 9.

Литература

1. Расторгуев В.Я. Разработка учебной системы программирования. Компилятор с языка высокого уровня. [Электронный ресурс : учебное пособие], СПбПУ, СПб, 2012

Приложение 1

Санкт-Петербургский политехнический университет
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Разработка учебной системы программирования

Дисциплина: «Системы программирования»

Вариант № XX

Исполнители:

Студенты группы ……. Иванов И.И.

Петров П.П.

Руководитель Расторгуев В.Я.

Санкт-Петербург

2020

Приложение 2

Санкт-Петербургский политехнический университет
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Разработка учебной системы программирования
Компилятор с ЯВУ

Дисциплина: «Системы программирования»
Вариант № XX

Исполнители:

Студенты группы ……. Иванов И.И.

Петров П.П.

Руководитель Расторгуев В.Я.

Санкт-Петербург

2020

Приложение 3

1. Задание на курсовую работу

Задание на КР дано в следующей общей формулировке:

«Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте».

Общая формулировка персонализирована в варианте № XX, приведенном на рис. 1.

Условный персональный вариант № XX

```
EXAMP1: PROC OPTIONS ( MAIN );
        DCL A BIN FIXED ( 31 ) INIT ( 111B );
        DCL B BIN FIXED ( 31 ) INIT ( 100B );
        DCL C BIN FIXED ( 31 ) INIT ( 101B );
        DCL D BIN FIXED ( 31 );
        DCL E BIN FIXED ( 31 );
        D = A + B - C;
        E = D * A / B;
END EXAMP1;
```

Рис. 1

Схема функционирования макета учебной системы программирования, являющегося, согласно заданию на КР, одним из элементов исходных данных, приведена на рис. 2.

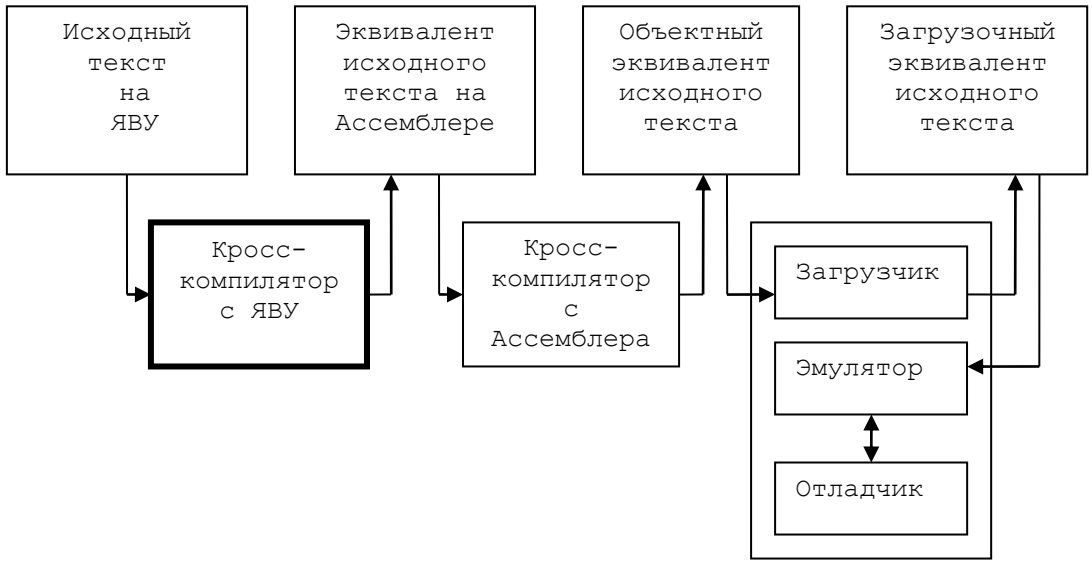

Рис. 2

Дополнительные требования к КР, вытекающие из рис.2, предполагают то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (IBM 370). 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 IBM 370, 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 IBM 370 и хранится в виде двоичного файла технологической ЭВМ с расширением *.tex;
- загрузочный эквивалент исходной программы представляет собой машинный код IBM 370, запоминаемый в области ОЗУ технологической ЭВМ, являющейся зоной загрузки для эмулятора объектной ЭВМ.

На рис. 2 жирным контуром выделен компилятор с ЯВУ, являющийся тем элементом макета, результаты модификации которого под персональный вариант № XX (см. рис 1) приведены в данном отчете.

Приложение 4

2. Постановка задачи модификации макета в части компилятора с ЯВУ

Схема функционирования модифицированного макетного компилятора с ЯВУ должна соответствовать рис. 3.

Рис. 3

Функционал модифицированного макетного компилятора должен обеспечить компиляцию исходного текста на ЯВУ (см. рис. 1) в эквивалент исходного текста на Ассемблере (см. рис. 4)

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14

RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 4

Согласованные функциональные ограничения

По согласованию с преподавателем в модифицированном макетном компиляторе введены следующие функциональные ограничения:

1) Компилятор игнорирует различия в приоритетах арифметических операций умножения «*», деления «/» и сложения «+», вычитания «-», считая все арифметические операции равноприоритетными.
2) . . .

Приложение 5

3. Модификация состава и структуры БД макета компилятора с ЯВУ

3.1 Модификация БД лексического анализатора

БД лексического анализатора определяется путем выделения подмножества терминальных символов, в которое входят все терминальные символы, являющиеся разделителями термов языка.
Для персонального варианта (см. рис. 1) разделителями термов являются терминальные символы из множества:

{ ":" , "(" , ")" , ";" , "+" , "-" ,"=" , "_" , **“*”** , **“/”** },

где модификация лексических настроек демо-примера состоит в расширении списка символов-разделителей термов дополнительными символами звездочка “*****” и слэш “/”

Т.к. авторы макета лексического анализатора разместили подмножество символов-разделителей термов прямо в программе лексического анализатора **compress_ISXTXT()**, а не в специальной таблице, то коррекция БД лексического анализатора выполнена в теле этой программы:

```
void compress_ISXTXT() {
. . .
/* Здесь размещаем коррекции с комментариями */
. . .
}
```

3.2 Модификация БД синтаксического анализатора

Высокоуровневое определение модифицированного синтаксиса для персонального варианта (см. рис. 1) с выделением модификаций жирным шрифтом в определение нетерминала <ZNK> приведен ниже:

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>);

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".

Сначала перепишем каждое правило грамматики, заменяя формат метаязыка Бэкуса на формат продукций (или на форму распознавания).
Для этого в каждой альтернативе каждого правила в формате метаязыка Бэкуса поменяем местами правую и левую части, изменив при этом символ "::=" на "⟵". Получим следующий набор продукций:

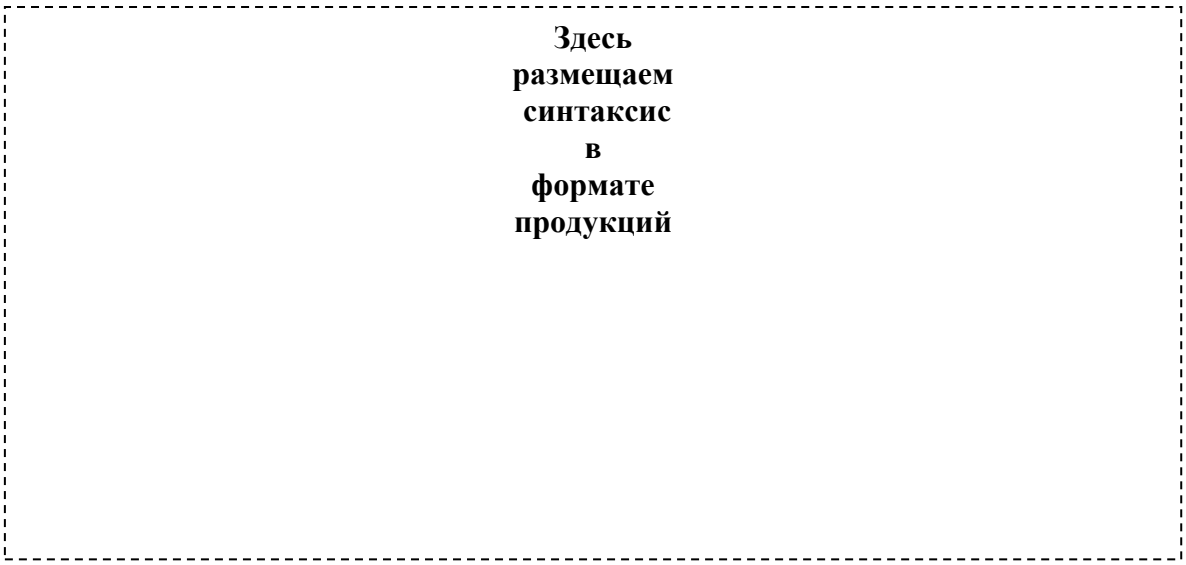

Рис. 5

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. Получим следующий результат:

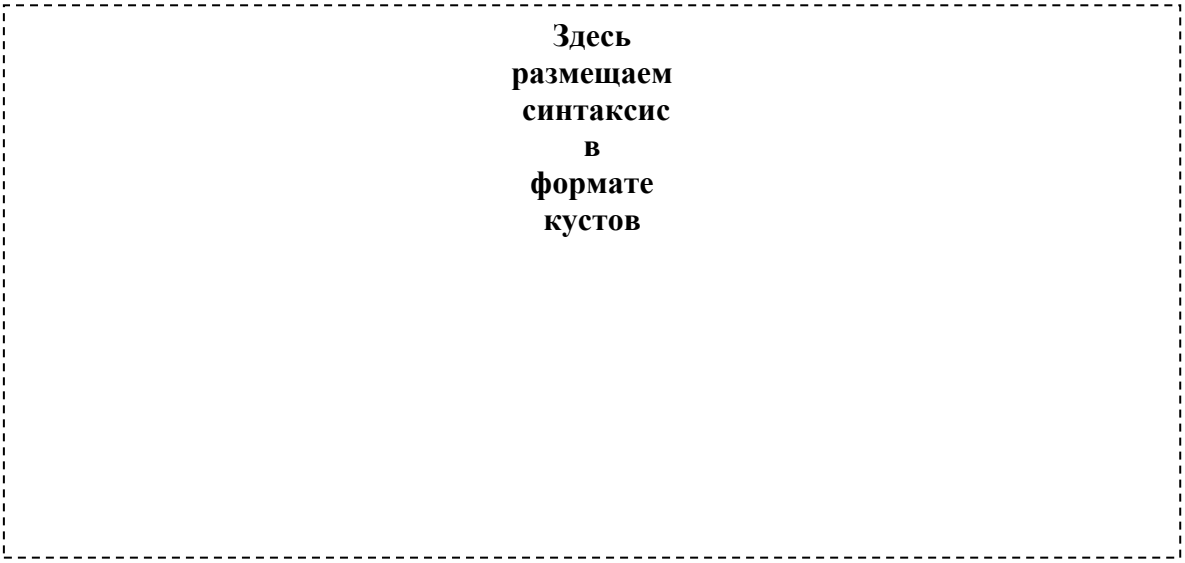
Здесь
размещаем
синтаксис
в
формате
кустов

Рис. 6

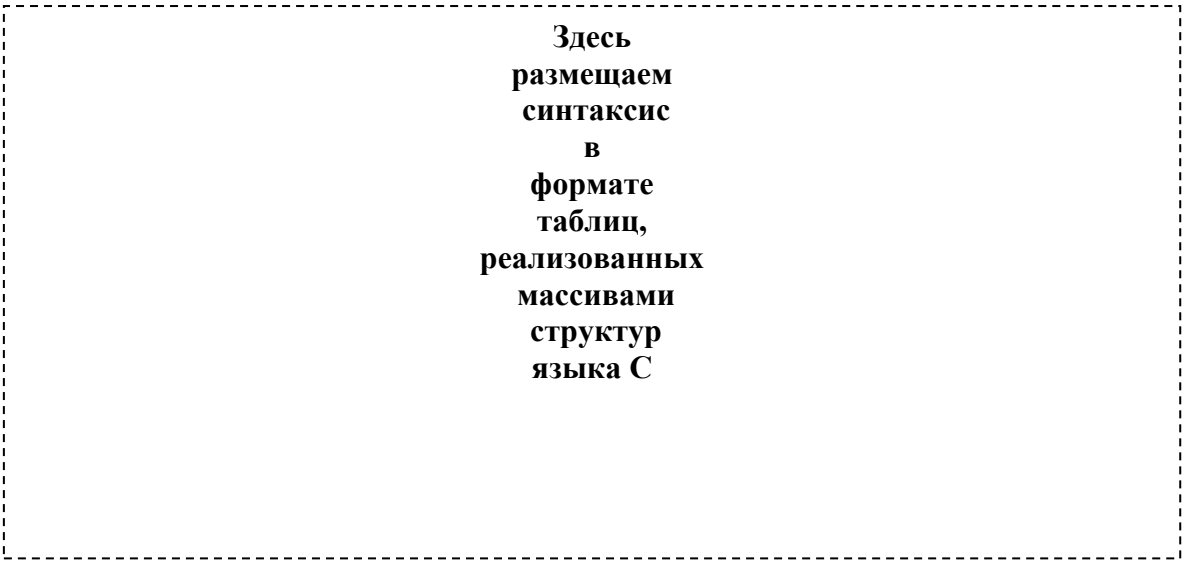
Здесь
размещаем
синтаксис
в
формате
таблиц,
реализованных
массивами
структур
языка С

Рис. 7

Приложение 6

4. Модификация алгоритма макета компилятора с ЯВУ

Модификация алгоритма макета компилятора с ЯВУ была произведена параметрически путем модификации той части параметров настройки компилятора ЯВУ, которая относится к семантике ЯВУ. Этой частью семантических параметров является библиотека семантических подпрограмм. Объем модификаций этой библиотеки в общем случае определяется объемом модификации синтаксиса ЯВУ. Для условного персонального варианта № XX модификация синтаксиса (см. раздел 3) привела к необходимости модификации алгоритма макета компилятора с ЯВУ в части семантической подпрограммы AVI2:

```
int AVI2 ()  {
.  .  .
/* здесь приведены модификации */
.  .  .
}
```

Приложение 7

5. Результаты экспериментальной проверки модифицированного компилятора с ЯВУ

Работоспособность модифицированного под персональный вариант макета компилятора с ЯВУ была подтверждена испытательным прогоном его в среде ОС Linux. В результате на выходе был получен текст на Ассемблере, который совпал с текстом, приведенным на рис. 4.

Приложение 8

6. Выводы по результатам работы

Порученное задание выполнено успешно. Успешность выполнения порученного задания подтверждена полным совпадением выходного текста, полученного при экспериментальной проверке компилятора с ЯВУ, с эталоном на рис. 4.

Приложение 9

Санкт-Петербургский политехнический университет
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Разработка учебной системы программирования

Дисциплина: «Системы программирования»

Вариант № XX

Исполнители:

Студенты группы ……. Иванов И.И.

Петров П.П.

Руководитель Расторгуев В.Я.

Санкт-Петербург

2020

Санкт-Петербургский политехнический университет
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Разработка учебной системы программирования
Компилятор с ЯВУ

Дисциплина: «Системы программирования»
Вариант № XX

Исполнители:

Студенты группы ……. Иванов И.И.

Петров П.П.

Руководитель Расторгуев В.Я.

Санкт-Петербург

2020

2. Задание на курсовую работу

Задание на КР дано в следующей общей формулировке:

«Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте».

Общая формулировка персонализирована в варианте № XX, приведенном на рис. 1.

Условный персональный вариант № XX

```
EXAMP1: PROC OPTIONS ( MAIN );
        DCL A BIN FIXED ( 31 ) INIT ( 111B );
        DCL B BIN FIXED ( 31 ) INIT ( 100B );
        DCL C BIN FIXED ( 31 ) INIT ( 101B );
        DCL D BIN FIXED ( 31 );
        DCL E BIN FIXED ( 31 );
        D = A + B - C;
        E = D * A / B;
END EXAMP1;
```

Рис. 1

Схема функционирования макета учебной системы программирования, являющегося, согласно заданию на КР, одним из элементов исходных данных, приведена на рис. 2.

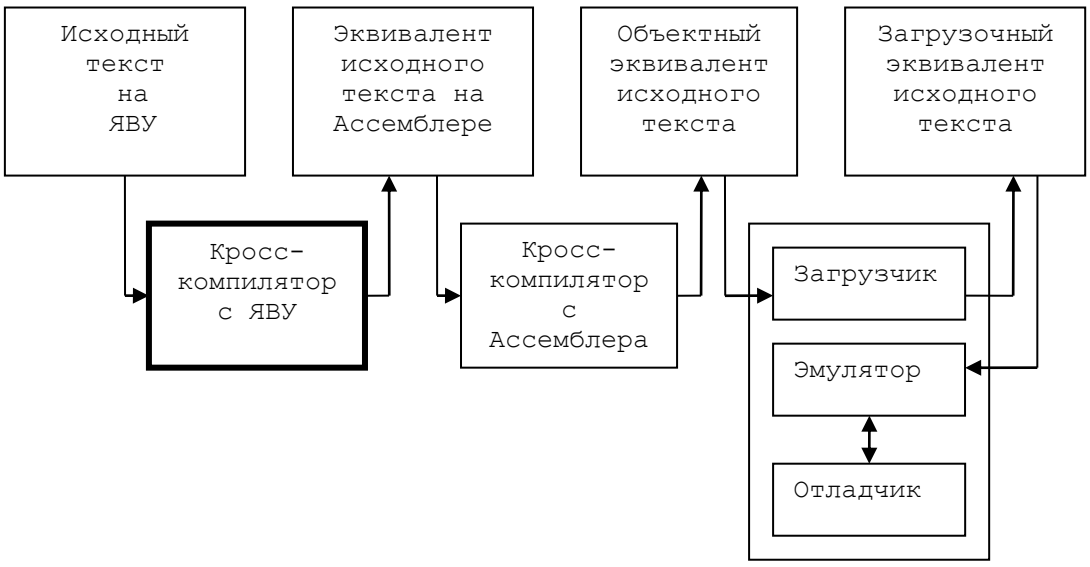

Рис. 2

Дополнительные требования к КР, вытекающие из рис.2, предполагают то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (IBM 370). 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 IBM 370, 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 IBM 370 и хранится в виде двоичного файла технологической ЭВМ с расширением *.tex;
- загрузочный эквивалент исходной программы представляет собой машинный код IBM 370, запоминаемый в области ОЗУ технологической ЭВМ, являющейся зоной загрузки для эмулятора объектной ЭВМ.

На рис. 2 жирным контуром выделен компилятор с ЯВУ, являющийся тем элементом макета, результаты модификации которого под персональный вариант № XX (см. рис 1) приведены в данном отчете.

2. Постановка задачи модификации макета в части компилятора с ЯВУ

Схема функционирования модифицированного макетного компилятора с ЯВУ должна соответствовать рис. 3.

Рис. 3

Функционал модифицированного макетного компилятора должен обеспечить компиляцию исходного текста на ЯВУ (см. рис. 1) в эквивалент исходного текста на Ассемблере (см. рис. 4)

Метка	КОП	Операнды	Пояснения
ex	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB

	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 4

Согласованные функциональные ограничения

По согласованию с преподавателем в модифицированном макетном компиляторе введены следующие функциональные ограничения:

3) Компилятор игнорирует различия в приоритетах арифметических операций умножения «*», деления «/» и сложения «+», вычитания «-», считая все арифметические операции равноприоритетными.
4) . . .

3. Модификация состава и структуры БД макета компилятора с ЯВУ

3.1 Модификация БД лексического анализатора

БД лексического анализатора определяется путем выделения подмножества терминальных символов, в которое входят все терминальные символы, являющиеся разделителями термов языка.
Для персонального варианта (см. рис. 1) разделителями термов являются терминальные символы из множества:

{ ":" , "(" , ")" , ";" , "+" , "-" ,"=" , "_" , “*” , “/” },

где модификация лексических настроек демо-примера состоит в расширении списка символов-разделителей термов дополнительными символами звездочка “**”** и слэш “/”

Т.к. авторы макета лексического анализатора разместили подмножество символов-разделителей термов прямо в программе лексического анализатора **compress_ISXTXT()**, а не в специальной таблице, то коррекция БД лексического анализатора выполнена в теле этой программы:

```
void compress_ISXTXT() {
. . .
/* Здесь размещаем коррекции с комментариями */
. . .
}
```

3.2 Модификация БД синтаксического анализатора

Высокоуровневое определение модифицированного синтаксиса для персонального варианта (см. рис. 1) с выделением модификаций жирным шрифтом в определение нетерминала <ZNK> приведен ниже:

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>);
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".

Сначала перепишем каждое правило грамматики, заменяя формат метаязыка Бэкуса на формат продукций (или на форму распознавания).
Для этого в каждой альтернативе каждого правила в формате метаязыка Бэкуса поменяем местами правую и левую части, изменив при этом символ "::=" на "⟶". Получим следующий набор продукций:

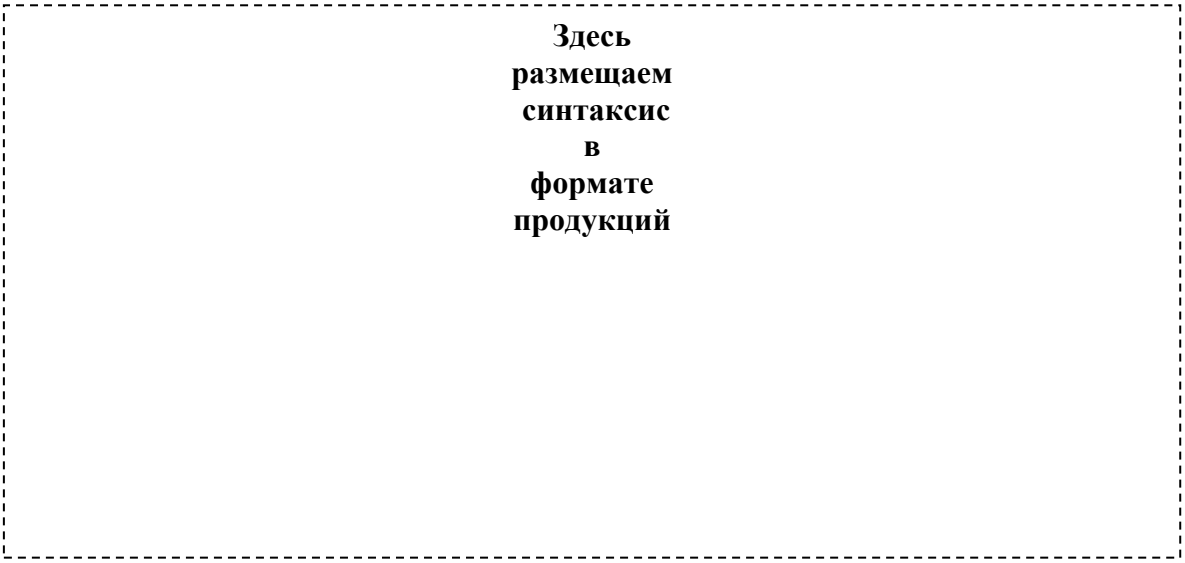

Рис. 5

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. Получим следующий результат:

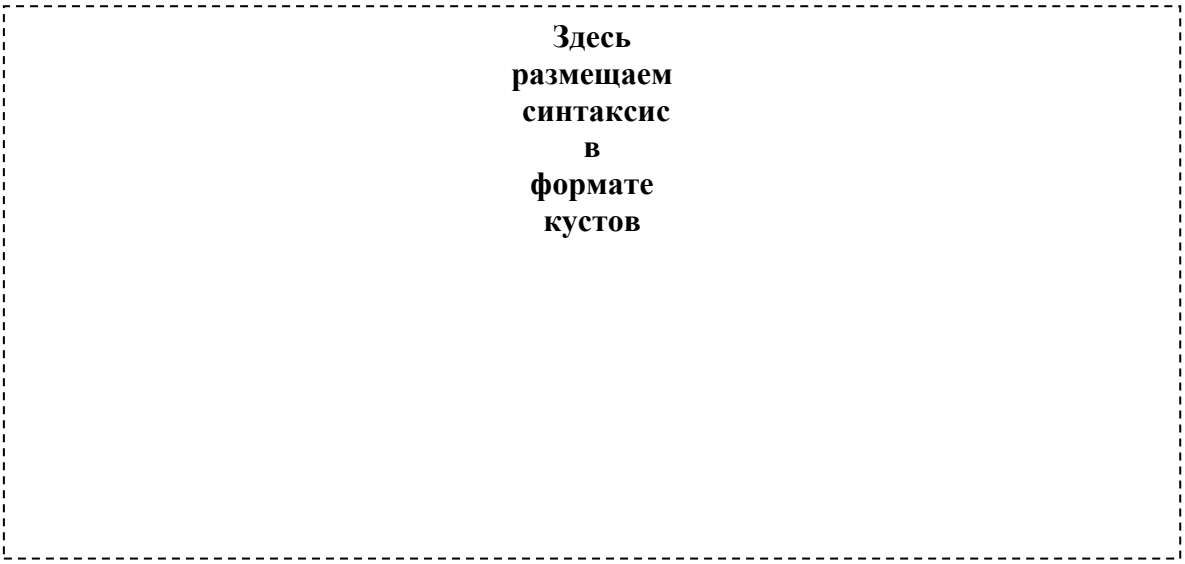

Рис. 6

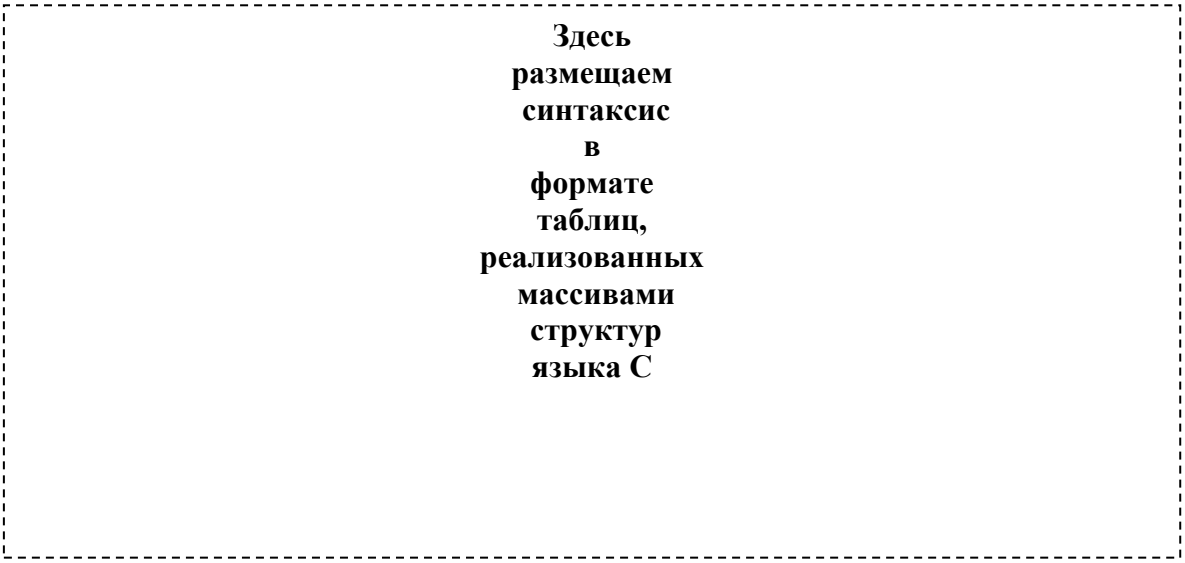

Рис. 7

4. Модификация алгоритма макета компилятора с ЯВУ

Модификация алгоритма макета компилятора с ЯВУ была произведена параметрически путем модификации той части параметров настройки компилятора ЯВУ, которая относится к семантике ЯВУ. Этой частью семантических параметров является библиотека семантических подпрограмм. Объем модификаций этой библиотеки в общем случае определяется объемом модификации синтаксиса ЯВУ. Для условного персонального варианта № XX модификация синтаксиса (см. раздел 3) привела к необходимости модификации алгоритма макета компилятора с ЯВУ в части семантической подпрограммы AVI2:

```
int AVI2 ()  {
.  .  .
/* здесь приведены модификации */
.  .  .
}
```

5. Результаты экспериментальной проверки модифицированного компилятора с ЯВУ

Работоспособность модифицированного под персональный вариант макета компилятора с ЯВУ была подтверждена испытательным прогоном его в среде ОС Linux. В результате на выходе был получен текст на Ассемблере, который совпал с текстом, приведенным на рис. 4.

6. Выводы по результатам работы

Порученное задание выполнено успешно. Успешность выполнения порученного задания подтверждена полным совпадением выходного текста, полученного при экспериментальной проверке компилятора с ЯВУ, с эталоном на рис. 4.

Низкоуровневый эквивалент построен и согласован. С чего начинать разработку компилятора с Ассемблера?

Чтобы ответить на вопрос, поставленный в заголовке текста, сделаем два действия:

1) Посмотрим на формулировку персонального задания, которая одинакова для всех вариантов и размещена в начале перечня всех вариантов персональных заданий, содержащегося в файле variantn.rtf внутри архива с исходными данными для КР, переданного в группы в начале семестра. Вот она:

«**Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте**».

2) Для того, чтобы полнее раскрыть смысл лаконичной формулировки персонального задания, приведенной выше, вспомним про учебное пособие [1] (далее в тексте «**Пособие**»), доступного в библиотеке СПбПУ и также переданного в группы внутри архива с исходными данными для КР, раскроем его, найдем в оглавлении **Пособия** раздел «**Рекомендации по порядку выполнения работы**» и прочитаем следующее:

«**. . .**

Данная лабораторная работа предполагает использование в качестве начальной основы отлаженной и работающей программы-макета компилятора с АССЕМБЛЕРА IBM 370, с ограниченным набором языковых конструкций, достаточным для написания законченной правильной программы контрольного примера. Программа-макет компилятора с АССЕМБЛЕРА написана на алгоритмическом языке Си.

Студенту предлагается расширить первичный набор языковых конструкций контрольного примера до уровня, позволяющего оттранслировать без ошибок пример, содержащийся в его личном задании. При этом необходимо:

***а)** Ознакомиться с исходными текстами программы-макета и контрольного примера, на который настроена программа-макет компилятора с АССЕМБЛЕРА,*

***б)** Проверить работоспособность программы-макета компилятора с АССЕМБЛЕРА путем проведения компиляции контрольного примера,*

***в)** Вручную построить ассемблеровский эквивалент исходного текста примера, содержащегося в личном задании, и выделить из него множество языковых конструкций, отсутствующих в ассемблеровском эквиваленте исходного текста контрольного примера,*

***г)** Спроектировать перечень модификаций элементов БД и подпрограмм программы-макета компилятора с АССЕМБЛЕРА с целью обеспечения компиляции ассемблеровского эквивалента примера из личного задания,*

***д)** Ввести модификации в исходный текст программы-макета компилятора с АССЕМБЛЕРА, снабдив измененные места исходного текста построчным комментарием, и записать модифицированный вариант в свой личный директорий на сервере,*

е) *Выполнить отладку модифицированной программы-макета компилятора с АССЕМБЛЕРА,*

ж) *Выполнить отработку своего ассемблеровского эквивалента личного задания с помощью модифицированной программы-макета компилятора с АССЕМБЛЕРА,*

з) *Составить отчет по результатам работы.*

. . .»

Переходим к выполнению рекомендаций из пп. от **а)** до **з).**

Действия по пункту а)

В архиве с исходными данными для КР находим папку:

StudLabRab_2020_magistry\LinuxLabRab\LinuxLabRab.TGZ\LinuxLabRab.tar\.\linux_lab_rab

Убеждаемся, во-первых, в том, что в этой папке имеются С-файлы: **Komppl.c** и **Kompassr.c.** Делая беглый входной контроль содержимого файлов: **Komppl.c** и **Kompassr.c**, убеждается в том, что они и есть исходные тексты для, соответственно, С-программы макета компилятора с ЯВУ и С-программы с языка Ассемблера, которык должны быть доработаны исполнителями КР под свой персональный вариант задания. Далее находим PL/1-файл **examppl.pli** контрольного или, иначе, демо-примера и знакомимся с ним:

```
EXAMP1: PROC OPTIONS ( MAIN );

      DCL A BIN FIXED ( 31 ) INIT ( 111B );
      DCL B BIN FIXED ( 31 ) INIT ( 100B );
      DCL C BIN FIXED ( 31 ) INIT ( 101B );
      DCL D BIN FIXED ( 31 );

      D = A + B - C;

END EXAMP1;
```

Рис.1

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту б)

Устанавливаем на компьютере ОС Linux для 32-хразрядной IBM PC. Переписываем туда архив с исходными данными для КР, включая папку **linux_lab_rab** .

Выполняем те две части shell-скрипта **GenSysProg** (генерация системы программирования), которые относятся к построению, соответственно, из исходного С-файла **Komppl.c** исполняемого файла **Komppl.exe**, а из исходного файла **Kompassr.c** исполняемого файла **Kompassr.exe**/

Выполняем те две части shell-скрипта **StartTestTask**, которые относятся к запуску компилятора с ЯВУ (т.е. файла **Komppl.exe)** для компиляции демо-примера (т.е. файла **exappl.pli**) с получением эквивалента на Ассемблере **examppl.ass** и, далее, объектного эквивалента **examppl.tex**

Убеждаемся в том, что эквивалент демо-примера на Ассемблере **examppl.ass** построен компилятором успешно и что он содержит следующий текст на Ассемблере:

Метка	**КОП**	**Операнды**	**Пояснения**
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
	END		Конец текста блока

Рис.2

Убеждаемся в том, что объектный эквивалент **examppl.tex** построен и содержит следующий набор объектных карт:

******************** Начало пакета объектных карт *********************

	Тип карты		Имя	Тип им.	Отн. адрес	Длина	
	‘ESD’		**EXAMP1**	**0**	**0x00**	**0x24**	

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x00**	**2**	**05 50**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x02**	**4**	**58 30 50 12**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x06**	**4**	**5A 30 50 16**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0A**	**4**	**5B 30 50 1A**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 1E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x12**	**2**	**07 FE**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x14**	**4**	**00 00 00 03**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x18**	**4**	**00 00 00 04**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1C**	**4**	**00 00 00 05**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x20**	**4**	**XX XX XX XX**

Тип карты	
‘END’	

********** **Конец пакета объектных карт** **********

Рис. 3

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту в)

Продолжение настоящих рекомендаций основано на специально построенном для этой цели условном варианте персонального задания, приведенном ниже и имеющем в своей основе демо-пример, приведенный выше (см. рис. 1):

Условный персональный вариант

```
EXAMP1: PROC OPTIONS ( MAIN );

      DCL A BIN FIXED ( 31 ) INIT ( 111B );
      DCL B BIN FIXED ( 31 ) INIT ( 100B );
      DCL C BIN FIXED ( 31 ) INIT ( 101B );
      DCL D BIN FIXED ( 31 );
      DCL E BIN FIXED ( 31 );
```

```
      D = A + B - C;
      E = D * A / B;

END EXAMP1;
```

Рис. 4

При визуальном сравнении исходных текстов условного персонального задания (см. рис. 4) и демо-примера (см. рис. 1) отмечаем то, что отличия в них минимальны и состоят в расширении:

- терминального алфавита грамматики ЯВУ PL/1 путем введения в него новых терминальных символов «*****» и «/»,
- перечня допустимых арифметических операций, применяемых в арифметическом выражении, находящемся в правой части оператора присваивания, операцией **умножения «*»** и операцией **деления «/»**.

Понятно, что отличия, обнаруженные в исходных текстах условного персонального задания и демо-примера, обусловят отличия их эквивалентов на языке Ассемблер. На рис. 5 приведен один из возможных вариантов такого эквивалента для условного персонального задания, приведенного на рис. 4.

Метка	**КОП**	**Операнды**	**Пояснения**
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
RRAB1	EQU	6	RRAB1 назначим 6

	END		Конец текста блока

Рис. 5

Для определения объема приращения функционала макетного варианта компилятора с Ассемблера, исходно рассчитанного только для обработки эквивалента демо-примера (см. рис. 2), с целью обеспечения возможности компиляции эквивалента на Ассемблере для условного персонального задания (см. рис. 5) нужно сравнить эти тексты и выделить строки с «новым» синтаксисом. Сделав это, убедимся, что к таким строкам следует отнести строки ассемблеровского эквивалента персонального задания с операциями умножения (**M**) и деления (**D**), которые имеют следующий синтаксис:

Метка	**КОП**	**Операнды**	**Пояснения**
	M	RRAB1,A	
	D	RRAB1,B	

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту г)

В основу функционирования макета компилятора с Ассемблера положен (см. **Пособие**) алгоритм, использующий в своей работе, во-первых, БД, включающую в себя следующие элементы:

- исходный текст,
- счетчик адреса (**Сч.Адр**.),
- таблица символов (**TSYM**),
- таблица псевдоопераций (**TPOP**),
- таблица машинных операций (**TMOP**),
- таблица базовых адресов (**TBASE**),
- двоичный эквивалент исходного текста,

и, во-вторых, набор специализированных функциональных блоков, которые могут быть разделены на две группы:

- функциональные блоки обработки ассемблеровских операторов, соответствующих машинным командам. В макетном варианте компилятора с Ассемблера имеется только два блока: для машинных команд **RR** и **RX** типа,
- функциональные блоки обработки ассемблеровских псевдооператоров, к числу которых в макетном варианте компилятора с Ассемблера следует отнести блоки обработки для псевдоопераций: **START**, **DC**, **DS**, **USING**, **EQU**, **END**.

Такая структуризация алгоритма макетного компилятора с Ассемблера облегчает поиск точек модификации для наращивания его функционала в интересах своего варианта задания.
Так, например, как установлено выше, для условного персонального задания необходимо «научить» макетный компилятор обрабатывать ассемблеровские операторы **M** и **D**, соответствующие машинным командам **умножения** и **деления**, обе из которых имеют тип **RX**. Для этого потребуется всего лишь дополнить таблицу машинных операций **TMOP**

двумя строками с характеристиками команд: **M** и **D**. Доработка новых функциональных блоков не потребуется.

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту д)

Методика предварительной подготовки и ввода коррекций в исходный текст программы-макета **Kompassr.c** описана в **Пособие.**

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту е) и ж)

Отладка программы макета компилятора с Ассемблера, модифицированного правками, производится студентами с опорой на собственный опыт написания и отладки Linux-приложений.
Отладку можно считать законченной в случае, если в процессе экспериментальной проверки работоспособности после запуска отлаживаемой программы компилятора с Ассемблера на выходе генерируется объектный эквивалент входного персонального задания, который совпадает с эталонным вариантом, согласованным с преподавателем на этапе постановки задачи (в ТЗ). В случае условного персонального варианта задания вход соответствует рис. 5, а выход рис. 6.
Работа отлаженной версии компилятора предъявляется (демонстрируется) преподавателю с одновременным вручением отчета для проверки/принятия.

********************** Начало пакета объектных карт *************************

	Тип карты		Имя	Тип им.	Отн. адрес	Длина	
	‘ESD’		**EXAMP1**	**0**	**0x00**	**0x38**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x00**		**2**		**05 50**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x02**		**4**		**58 30 50 22**	

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x06**	**4**	**5A 30 50 26**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0A**	**4**	**5B 30 50 2A**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x12**	**4**	**58 70 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x16**	**4**	**5C 60 50 22**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1A**	**4**	**5D 60 50 24**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1E**	**4**	**50 70 50 32**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x22**	**2**	**07FE**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x24**	**4**	**00 00 00 03**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x28**	**4**	**00 00 00 04**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x2C**	**4**	**00 00 00 05**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x30**	**4**	**XX XX XX XX**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x34**	**4**	**XX XX XX XX**

Тип карты	
‘END’	

********** **Конец пакета объектных карт** **********

Рис. 6

В крайнем случае, если появились проблемы и/или вопросы, то идем с ними за как в консультацией к преподавателю.

Действия по пункту з)

Отчет по результатам составляется с учетом «**Требований к содержанию отчета**», приведенных, как в разделе 4 **Пособия**, так и ниже:
« **. . .**

- описание примера из личного задания, включая его и пояснения модификаций состава и (или) структуры ассемблеровский эквивалент (указать формулировку личного задания),
- постановка задачи модификации программы-макета компилятора с АССЕМБЛЕРА (описание выбранного способа модификации программы-макета),
- перечень БД программы-макета компилятора с АССЕМБЛЕРА,
- перечень и пояснения модификаций алгоритма программы-макета компилятора с АССЕМБЛЕРА (допустимыми являются описания в виде: блок-схемы, текста на Си, а в очевидных случаях обычным текстом),
- выводы по результатам работы (указать позитивные и негативные моменты, проявившиеся в процессе работы, с общей личной оценкой проделанной работы).

. . . »

Если появились проблемы и/или вопросы, то идем с ними за как в консультацией к преподавателю.

Литература

1. Расторгуев В.Я. Разработка элементов учебной системы программирования. Компилятор с Ассемблера [электронный ресурс: учебное пособие], СПбПУ, Спб., 2012

Как оформить отчет по результатам разработки компилятора с Ассемблера?

Титульный лист на элемент системы

Так как под элементом системы программирования здесь понимается компилятор с Ассемблера, то частный титульный лист для этой части КР следует оформлять, согласно Приложению1.

Перечень разделов отчета

Согласно разделу 4 «**Требований к содержанию отчета**» из методического пособия [1], в материал отчета следует включить:

« . . .

- описание примера из личного задания, включая его самого и пояснения модификаций состава и (или) структуры его ассемблеровского эквивалента (указать формулировку личного задания),
- постановка задачи модификации программы-макета компилятора с АССЕМБЛЕРА (описание выбранного способа модификации программы-макета),
- перечень модификаций БД программы-макета компилятора с АССЕМБЛЕРА,
- перечень и пояснения модификаций алгоритма программы-макета компилятора с АССЕМБЛЕРА (допустимыми являются описания в виде: блок-схемы, текста на Си, а в очевидных случаях обычным текстом),
- Результаты экспериментальной проверки модифицированного макета компилятора с Ассемблера,
- выводы по результатам работы (указать позитивные и негативные моменты, проявившиеся в процессе работы, с общей личной оценкой проделанной работы).

. . .»

Далее будут приведены примеры выполнения требований к разделам отчета, основанные на условном персональном варианте задания, взятом из Рекомендаций_3, точнее говоря, на ассемблеровском эквиваленте условного персонального задания, который воспроизведен ниже (см. рис. 2):

Эквивалент условного персонального задания на Ассемблере

Метка	КОП	Операнды	Пояснения
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB

	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F'3'	A=3
B	DC	F'4'	B=4
C	DC	F'5'	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 2

Пояснения к разделу «Задание на КР»

В этом разделе отчета логично дать формулировку всего задания на КР, уточнить свой персональный вариант и далее сфокусировать внимание на тех его аспектах, результаты работы над которыми войдут в настоящий отчет.
Пример реализации раздела отчета приведен в приложении 2.

Пояснения к разделу «Постановка задачи модификации макета в части компилятора с Ассемблера»

В этом разделе следует привести постановку задачи (техническое задание) на ту часть КР, которая документируется данным отчетом. В данном случае это компилятор с Ассемблера, который должен быть получен путем модификации функционала макетного образца компилятора с Ассемблера под персональный вариант задания.
Понятно, что, согласно существующей практике, черновик ТЗ составляет исполнитель (студент), после чего согласует его с преподавателем, далее приступает к его реализации и переносит его в отчет. В ТЗ должны быть определены:

- предмет разработки,
- требования по входу и выходу,
- опционно, при согласовании могут быть предложены Заказчику (преподавателю) и согласованы с ним мотивированные ограничения функционала предмета разработки.

Пример реализации раздела отчета приведен в приложении 3.

Пояснения к разделу «Модификация состава и структуры БД макета компилятора с Ассемблера»

Так как алгоритм функционирования макетного компилятора с Ассемблера в своей работе опирается (см. [1]) на БД, включающую в себя следующие элементы:

- исходный текст,
- счетчик адреса (**Сч.Адр**.),
- таблица символов (**TSYM**),
- таблица псевдоопераций (**TPOP**),
- таблица машинных операций (**TMOP**),
- таблица базовых адресов (**TBASE**),
- двоичный эквивалент исходного текста,

то в данном разделе отчета следует конкретно указать элементы БД, подлежащие модификации в интересах поддержки компиляции ассемблеровского эквивалента условного персонального задания (см. рис. 2).
Пример реализации раздела отчета приведен в приложении 4.

Пояснения к разделу «Модификация алгоритма макета компилятора с Ассемблера

Так как алгоритм функционирования макетного компилятора с Ассемблера реализован в виде набора специализированных функциональных блоков, которые могут быть разделены на две группы:

- функциональные блоки обработки ассемблеровских операторов, соответствующих машинным командам. В макетном варианте компилятора с Ассемблера имеется только два блока: для машинных команд **RR** и **RX** типа,
- функциональные блоки обработки ассемблеровских псевдооператоров, к числу которых в макетном варианте компилятора с Ассемблера следует отнести блоки обработки для псевдоопераций: **START**, **DC**, **DS**, **USING**, **EQU**, **END**.

то в отчете следует указать только те из них, которые нуждаются в модификации в интересах поддержки условного персонального задания.

Пример реализации раздела отчета приведен в приложении 5.

Пояснения к разделу «Результаты экспериментальной проверки модифицированного компилятора с Ассемблера»

После отладки модифицированного макета до состояния, когда на выходе модифицированного компилятора с Ассемблера получится объектный эквивалент персонального варианта на Ассемблере, приведенного на рис. 2, пишем этот раздел отчета.

Пример реализации этого раздела приведен в приложении 6.

Пояснения к разделу «Выводы по результатам работы»

В начале раздела необходимо разместить две обязательные фразы.

Первая фраза отражает самооценку разработчика в одном из двух вариантов:

1) Порученное задание мною выполнено успешно.
2) Порученное задание мною не выполнено.

Вторая фраза доказательно подтверждает первую фразу в одном из двух возможных вариантов:

1) Успешность выполнения порученного задания подтверждена полным совпадением выходного объектного текста, полученного при экспериментальной проверке компилятора с Ассемблера, с эталоном.
2) Неудача выполнения порученного задания подтверждена несовпадением выходного объектного текста, полученного при экспериментальной проверке компилятора с Ассемблера, с эталоном.

Третья и последующие фразы добавляются исполнителем опционно и без особой надежды на то, что эти фразы будут внимательно прочитаны Заказчиком.

Пример реализации этого раздела приведен в приложении 7.

Пояснения ко всему отчету в целом

Пример целого отчета, в котором собраны все формулировки разделов, рекомендованные выше, приведен в приложении 8.

Литература

1. Расторгуев В.Я. Разработка учебной системы программирования. Компилятор с языка Ассемблера. [Электронный ресурс : учебное пособие], СПбПУ, СПб, 2012

Приложение 1

Санкт-Петербургский политехнический университет
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Разработка учебной системы программирования
Компилятор с Ассемблера

Дисциплина: «Системы программирования»
Вариант № XX

Исполнители:

Студенты группы ……. Иванов И.И.

Петров П.П.

Руководитель Расторгуев В.Я.

Санкт-Петербург

2020

Приложение 2

1. Задание на курсовую работу

Задание на КР дано в следующей общей формулировке:

«Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте».

Общая формулировка персонализирована в варианте № XX, приведенном на рис. 1.

Условный персональный вариант № XX

```
EXAMP1: PROC OPTIONS ( MAIN );
        DCL A BIN FIXED ( 31 ) INIT ( 111B );
        DCL B BIN FIXED ( 31 ) INIT ( 100B );
        DCL C BIN FIXED ( 31 ) INIT ( 101B );
        DCL D BIN FIXED ( 31 );
        DCL E BIN FIXED ( 31 );
        D = A + B - C;
        E = D * A / B;
END EXAMP1;
```

Рис. 1

Схема функционирования макета учебной системы программирования, являющегося, согласно заданию на КР, одним из элементов исходных данных, приведена на рис. 2.

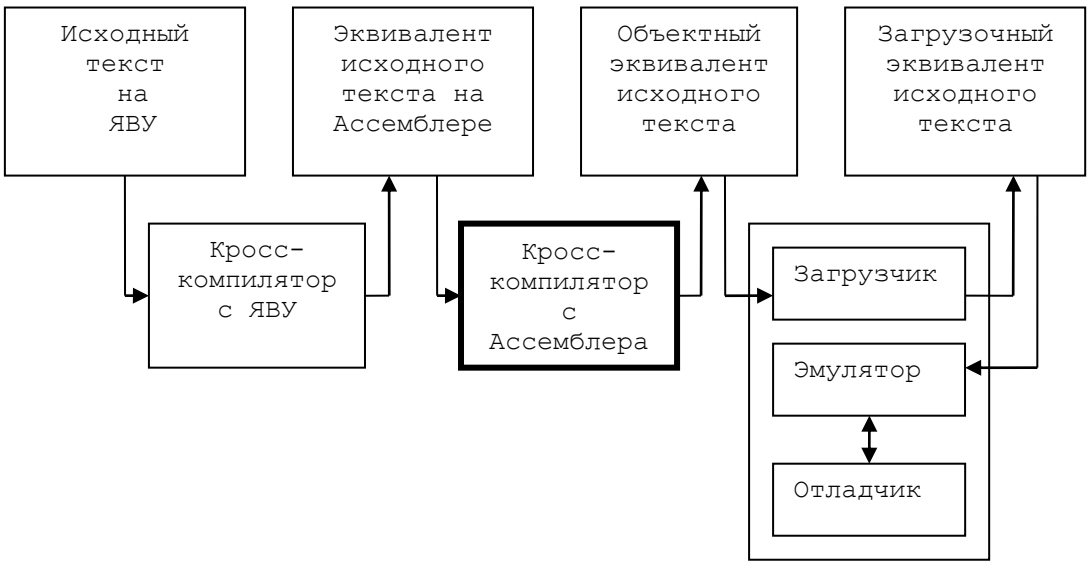

Рис. 2

Дополнительные требования к КР, вытекающие из рис.2, предполагают то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (IBM 370). 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 IBM 370, 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 IBM 370 и хранится в виде двоичного файла технологической ЭВМ с расширением *.tex;
- загрузочный эквивалент исходной программы представляет собой машинный код IBM 370, запоминаемый в области ОЗУ технологической ЭВМ, являющейся зоной загрузки для эмулятора объектной ЭВМ.

На рис. 2 жирным контуром выделен компилятор с Ассемблера, являющийся тем элементом макета, результаты модификации функционала которого под персональный вариант № XX (см. рис 1) приведены в данном отчете.

Согласно проектным решениям, реализованным при разработке компилятора с ЯВУ, после компиляции условного персонального задания (см. рис. 1) должен быть получен следующий его эквивалент на Ассемблере (см. рис. 3):

Метка	**КОП**	**Операнды**	**Пояснения**
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5

RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 3

Согласно рис. 2, этот текст на Ассемблере должен быть входом для компилятора с Ассемблера, являющегося предметом разработки, результат которой приведен в настоящем отчете.

Приложение 3

2. Постановка задачи модификации макета в части компилятора с Ассемблера

Схема функционирования модифицированного макетного компилятора с Ассемблера должна соответствовать рис. 4.

Рис. 4

Функционал модифицированного макетного компилятора должен обеспечить компиляцию исходного текста на Ассемблере (см. рис. 3) в его объектный эквивалент (см. рис. 5):

********************** Начало пакета объектных карт *************************

	Тип карты		Имя	Тип им.	Отн. адрес	Длина	
	'ESD'		**EXAMP1**	**0**	**0x00**	**0x38**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	'TXT'		**0x00**		**2**		**05 50**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	'TXT'		**0x02**		**4**		**58 30 50 22**	

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x06**	**4**	**5A 30 50 26**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0A**	**4**	**5B 30 50 2A**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x12**	**4**	**58 70 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x16**	**4**	**5C 60 50 22**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1A**	**4**	**5D 60 50 24**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1E**	**4**	**50 70 50 32**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x22**	**2**	**07FE**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x24**	**4**	**00 00 00 03**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x28**	**4**	**00 00 00 04**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x2C**	**4**	**00 00 00 05**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x30**	**4**	**XX XX XX XX**

	Тип карты		адр		дл		Фрагмент двоичного кода	
	‘**TXT**’		**0x34**		**4**		**XX XX XX XX**	

	Тип карты		
	‘**END**’		

********** **Конец пакета объектных карт** **********

Рис. 5.

Согласованные функциональные ограничения

По согласованию с преподавателем в модифицированном макетном компиляторе с Ассемблера введены следующие функциональные ограничения:

1) . . .
2) . . .

3. Модификация состава и структуры БД макета компилятора с Ассемблера

Сопоставление ассемблеровских эквивалентов демо-примера и условного персонального задания показало, что, что для поддержки последнего необходимо модифицировать макетный компилятор с Ассемблера, вводя возможность использования в ассемблеровских текстах новых машинных команд: **умножения (M)** и **деления (D)**. Для этого из всего множества элементов БД необходимо модифицировать только таблицу допустимых машинных команд **TMOP** (см. рис. 6), добавив две новые строки для команд **M** и **D** (см. рис. 7), которые на рис. 7 выделены жирным шрифтом.

Мнемоника	**КОП**	**Длина в байтах**	**Т**ип
BALR	0x05	2	RR
L	0x58	4	RX
A	0x5A	4	RX
S	0x5B	4	RX
ST	0x50	4	RX
BCR	0x07	2	RR

Рис. 6. TMOP – таблица машинных операций (команд) для демо-примера

Мнемоника	**КОП**	**Длина в байтах**	**Т**ип
BALR	0x05	2	RR
L	0x58	4	RX
A	0x5A	4	RX
S	0x5B	4	RX
ST	0x50	4	RX
M	**0x5C**	**4**	**RX**
D	**0x5D**	**4**	**RX**
BCR	0x07	2	RR

Рис. 7. TMOP – таблица машинных операций (команд) для условного персонального задания

Приложение 5

4. Модификация алгоритма макета компилятора с Ассемблера

Реализация алгоритма функционирования макетного компилятора с Ассемблера структурирована так, что обработка всех операторов языка Ассемблера, порождающих машинные команды одного и того же типа, выполняется одним и тем же функциональным блоком компилятора.
«Новые» для макетного компилятора операторы **умножения M** и **деления** D относятся к **RX**-типу, который уже реализован соответствующим функциональным **RX-блоком** этого же макетного компилятора. Поэтому, никаких дополнительных модификаций алгоритма макета компилятора с Ассемблера в интересах поддержки условного персонального задания делать не надо.

Приложение 6

5. Результаты экспериментальной проверки модифицированного компилятора с Ассемблера

Работоспособность модифицированного под персональный вариант задания макета компилятора с Ассемблера была подтверждена испытательным прогоном его в среде ОС Linux. В результате на выходе был получен объектный эквивалент компилируемого текста на Ассемблере (см. рис. 3), который совпал с объектным текстом, приведенным на рис. 5.

Приложение 7

6. Выводы по результатам работы

Порученное задание выполнено успешно. Успешность выполнения порученного задания подтверждена полным совпадением выходного объектного текста, полученного при экспериментальной проверке компилятора с Ассемблера, с эталоном на рис. 5.

Приложение 8

Санкт-Петербургский политехнический университет
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Разработка учебной системы программирования
Компилятор с Ассемблера

Дисциплина: «Системы программирования»
Вариант № XX

Исполнители:

Студенты группы ……. Иванов И.И.

Петров П.П.

Руководитель Расторгуев В.Я.

Санкт-Петербург

2020

2. Задание на курсовую работу

Задание на КР дано в следующей общей формулировке:

«Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте».

Общая формулировка персонализирована в варианте № XX, приведенном на рис. 1.

Условный персональный вариант № XX

```
EXAMP1: PROC OPTIONS ( MAIN );
        DCL A BIN FIXED ( 31 ) INIT ( 111B );
        DCL B BIN FIXED ( 31 ) INIT ( 100B );
        DCL C BIN FIXED ( 31 ) INIT ( 101B );
        DCL D BIN FIXED ( 31 );
        DCL E BIN FIXED ( 31 );
        D = A + B - C;
        E = D * A / B;
END EXAMP1;
```

Рис. 1

Схема функционирования макета учебной системы программирования, являющегося, согласно заданию на КР, одним из элементов исходных данных, приведена на рис. 2.

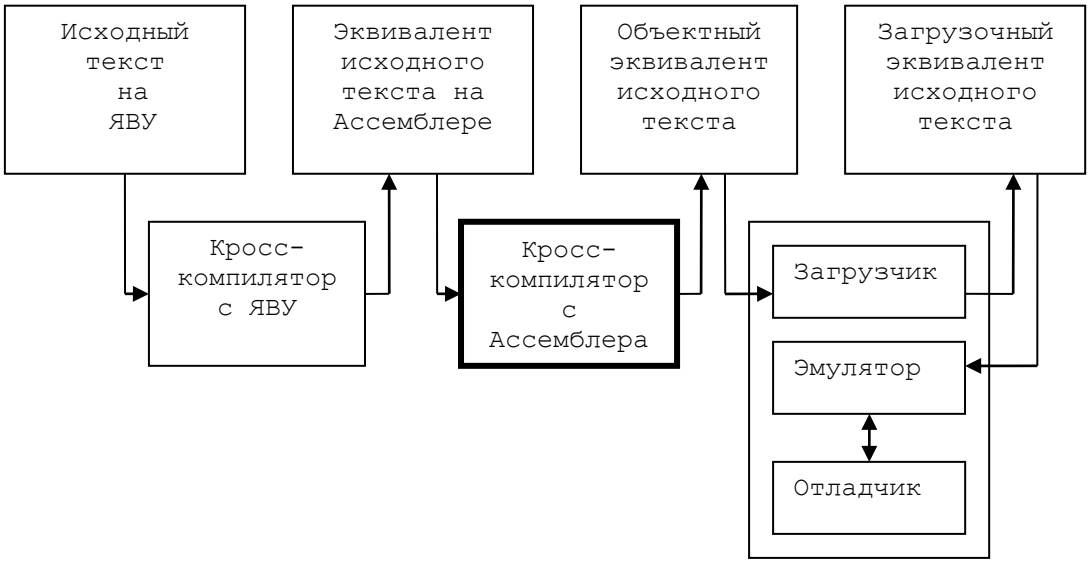

Рис. 2

Дополнительные требования к КР, вытекающие из рис.2, предполагают то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (IBM 370). 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 IBM 370, 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 IBM 370 и хранится в виде двоичного файла технологической ЭВМ с расширением *.tex;
- загрузочный эквивалент исходной программы представляет собой машинный код IBM 370, запоминаемый в области ОЗУ технологической ЭВМ, являющейся зоной загрузки для эмулятора объектной ЭВМ.

На рис. 2 жирным контуром выделен компилятор с Ассемблера, являющийся тем элементом макета, результаты модификации функционала которого под персональный вариант № XX (см. рис 1) приведены в данном отчете.

Согласно проектным решениям, реализованным при разработке компилятора с ЯВУ, после компиляции условного персонального задания (см. рис. 1) должен быть получен следующий его эквивалент на Ассемблере (см. рис. 3):

Метка	**КОП**	**Операнды**	**Пояснения**
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14

RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 3

Согласно рис. 2, этот текст на Ассемблере должен быть входом для компилятора с Ассемблера, являющегося предметом разработки, результат которой приведен в настоящем отчете.

2. Постановка задачи модификации макета в части компилятора с Ассемблера

Схема функционирования модифицированного макетного компилятора с Ассемблера должна соответствовать рис. 4.

Рис. 4

Функционал модифицированного макетного компилятора должен обеспечить компиляцию исходного текста на Ассемблере (см. рис. 3) в его объектный эквивалент (см. рис. 5):

********************** Начало пакета объектных карт *************************

	Тип карты		Имя	Тип им.	Отн. адрес	Длина	
	‘ESD’		**EXAMP1**	**0**	**0x00**	**0x38**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x00**		**2**		**05 50**	

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x02**	**4**	**58 30 50 22**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x06**	**4**	**5A 30 50 26**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0A**	**4**	**5B 30 50 2A**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x12**	**4**	**58 70 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x16**	**4**	**5C 60 50 22**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1A**	**4**	**5D 60 50 24**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1E**	**4**	**50 70 50 32**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x22**	**2**	**07FE**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x24**	**4**	**00 00 00 03**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x28**	**4**	**00 00 00 04**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x2C**	**4**	**00 00 00 05**

	Тип карты		адр		дл		Фрагмент двоичного кода	
	‘TXT’		**0x30**		**4**		**XX XX XX XX**	

	Тип карты		адр		дл		Фрагмент двоичного кода	
	‘TXT’		**0x34**		**4**		**XX XX XX XX**	

	Тип карты		
	‘END’		

********** **Конец пакета объектных карт** **********

Рис. 5.

Согласованные функциональные ограничения

По согласованию с преподавателем в модифицированном макетном компиляторе с Ассемблера введены следующие функциональные ограничения:

3) . . .
4) . . .

3. Модификация состава и структуры БД макета компилятора с Ассемблера

Сопоставление ассемблеровских эквивалентов демо-примера и условного персонального задания показало, что, что для поддержки последнего необходимо модифицировать макетный компилятор с Ассемблера, вводя возможность использования в ассемблеровских текстах новых машинных команд: **умножения (M)** и **деления (D)**. Для этого из всего множества элементов БД необходимо модифицировать только таблицу допустимых машинных команд **TMOP** (см. рис. 6), добавив две новые строки для команд **M** и **D** (см. рис. 7), которые на рис. 7 выделены жирным шрифтом.

Мнемоника	**КОП**	**Длина в байтах**	**Т**ип
BALR	0x05	2	RR

L	0x58	4	RX
A	0x5A	4	RX
S	0x5B	4	RX
ST	0x50	4	RX
BCR	0x07	2	RR

Рис. 6. ТМОР – таблица машинных операций (команд) для демо-примера

Мнемоника	**КОП**	**Длина в байтах**	**Т**ип
BALR	0x05	2	RR
L	0x58	4	RX
A	0x5A	4	RX
S	0x5B	4	RX
ST	0x50	4	RX
M	**0x5C**	**4**	**RX**
D	**0x5D**	**4**	**RX**
BCR	0x07	2	RR

Рис. 7. ТМОР – таблица машинных операций (команд) для условного персонального задания

4. Модификация алгоритма макета компилятора с Ассемблера

Реализация алгоритма функционирования макетного компилятора с Ассемблера структурирована так, что обработка всех операторов языка Ассемблера, порождающих машинные команды одного и того же типа, выполняется одним и тем же функциональным блоком компилятора.
«Новые» для макетного компилятора операторы **умножения M** и **деления** D относятся к **RX**-типу, который уже реализован соответствующим функциональным **RX-блоком** этого же макетного компилятора. Поэтому, никаких дополнительных модификаций алгоритма макета компилятора с Ассемблера в интересах поддержки условного персонального задания делать не надо.

5. Результаты экспериментальной проверки модифицированного компилятора с Ассемблера

Работоспособность модифицированного под персональный вариант задания макета компилятора с Ассемблера была подтверждена испытательным прогоном его в среде ОС Linux. В результате на выходе был получен объектный эквивалент компилируемого текста на Ассемблере (см. рис. 3), который совпал с объектным текстом, приведенным на рис. 5.

6. Выводы по результатам работы

Порученное задание выполнено успешно. Успешность выполнения порученного задания подтверждена полным совпадением выходного объектного текста, полученного при экспериментальной проверке компилятора с Ассемблера, с эталоном на рис. 5.

Объектный эквивалент на выходе компилятора с Ассемблера построен. Что делать дальше?

Чтобы ответить на вопрос, поставленный в заголовке текста, сделаем два действия:

1) Посмотрим на формулировку персонального задания, которая одинакова для всех вариантов и размещена в начале перечня всех вариантов персональных заданий, содержащегося в файле variantn.rtf внутри архива с исходными данными для КР, переданного в группы в начале семестра. Вот она:

«**Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте**».

2) Для того, чтобы полнее раскрыть смысл лаконичной формулировки персонального задания, приведенной выше, вспомним про учебное пособие [1] (далее в тексте «**Пособие**»), доступного в библиотеке СПбПУ и также переданного в группы внутри архива с исходными данными для КР, раскроем его, найдем в оглавлении **Пособия** раздел «**Рекомендации по порядку выполнения работы**» и прочитаем следующее:

«. . .
Данная лабораторная работа предполагает использование в качестве начальной основы отлаженной и работающей программы-макета, т.е. утилиты, содержащей:

- абсолютный загрузчик,
- эмулятор ЭВМ IBM 370,
- низкоуровневый отладчик программ, выполняющихся на эмуляторе.

По степени функциональности утилита способна демонстрировать работоспособность законченной правильной программы контрольного примера. Программа-макет написана на алгоритмическом языке Си и снабжена подробным контекстным комментарием.

Студенту предлагается, если необходимо, расширить первичный набор машинных операций эмулятора ЭВМ IBM 370, поддерживаемый утилитой, до уровня, позволяющего выполнить на эмуляторе пример, содержащийся в его личном задании, и произвести соответствующую доработку утилиты. При этом необходимо:

а) Ознакомиться с исходными текстами утилиты и контрольного примера, на который настроена утилита,

б) Проверить работоспособность утилиты путем проведения сборки из объектных модулей контрольного примера его исполняемого кода и запуска этого кода на выполнение с помощью эмулятора и отладчика утилиты,

в) Составить список машинных инструкций ЭВМ IBM 370, который необходим для выполнения примера, содержащегося в личном задании, и, одновременно, не поддержанный эмулятором,

г) Спроектировать перечень коррекций утилиты с целью реализации поддержки списка машинных инструкций ЭВМ IBM 370, составленный в предыдущем пункте,

д) Ввести коррекции в исходный текст утилиты, снабдив измененные места исходного текста построчным комментарием, и записать модифицированный вариант утилиты в свой личный директорий на сервере,

е) Выполнить отладку модифицированной утилиты,

ж) Провести экспериментальную проверку, подтверждающую работоспособность утилиты.

з) Составить отчет по результатам работы.

. . .»

Переходим к выполнению рекомендаций из пп. от **а)** до **з).**

Действия по пункту а)

В архиве с исходными данными для КР находим папку:

StudLabRab_2020_magistry\LinuxLabRab\LinuxLabRab.TGZ\LinuxLabRab.tar\.\linux_lab_rab

Убеждаемся, во-первых, в том, что в этой папке имеются С-файлы: **komppl.c**, **kompassr.c** и **absloadm.c.** Делая беглый входной контроль содержимого файлов: **komppl.c**, **kompassr.c** и **absloadm.c**, убеждается в том, что они и есть исходные тексты для, соответственно, С-программы макета компилятора с ЯВУ, С-программы компилятора с языка Ассемблера и С-программы для утилиты загрузчика, которые должны быть доработаны исполнителями КР под свой персональный вариант задания.
Далее находим PL/1-файл **examppl.pli** контрольного или, иначе, демо-примера и знакомимся с ним:

```
EXAMP1: PROC OPTIONS ( MAIN );

      DCL A BIN FIXED ( 31 ) INIT ( 11B );
      DCL B BIN FIXED ( 31 ) INIT ( 100B );
      DCL C BIN FIXED ( 31 ) INIT ( 101B );
      DCL D BIN FIXED ( 31 );

      D = A + B - C;

END EXAMP1;
```

Рис.1

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту б)

Устанавливаем на компьютере ОС Linux для 32-хразрядной IBM PC. Переписываем туда архив с исходными данными для КР, включая папку **linux_lab_rab** .

Выполняем shell-скрипт **GenSysProg** (генерация системы программирования), который относятся к построению, соответственно, из исходного С-файла **komppl.c** исполняемого файла **komppl.exe**, из исходного файла **kompassr.c** исполняемого файла **kompassr.exe** и из исходного С-файла **absloadm.c** исполняемого файла **absloadm.exe**.

Выполняем shell-скрипт **StartTestTask**, которые относятся к запуску компилятора с ЯВУ (т.е. файла k**omppl.exe)** для компиляции демо-примера (т.е. файла **exappl.pli**) с получением эквивалента на Ассемблере **examppl.ass**, далее файла **kompassr.exe** для компиляции **examppl.ass** и получения объектного эквивалента **examppl.tex** и, наконец, получения с помощью загрузчика исполняемого кода демо-примера с последующим исполнением его эмулятором.

Убеждаемся с помощью отладчика в том, что выполнение исполняемого кода демо-примера на эмуляторе происходит согласно его алгоритма.

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту в)

Продолжение настоящих рекомендаций основано на специально построенном для этой цели условном варианте персонального задания, приведенном ниже и имеющем в своей основе демо-пример, приведенный выше (см. рис. 1):

Условный персональный вариант

```
EXAMP1: PROC OPTIONS ( MAIN );

      DCL A BIN FIXED ( 31 ) INIT ( 11B );
      DCL B BIN FIXED ( 31 ) INIT ( 100B );
      DCL C BIN FIXED ( 31 ) INIT ( 101B );
      DCL D BIN FIXED ( 31 );
      DCL E BIN FIXED ( 31 );
      D = A + B - C;
      E = D * A / B;

END EXAMP1;
```

Рис. 2

При визуальном сравнении исходных текстов условного персонального задания (см. рис. 2) и демо-примера (см. рис. 1) отмечаем то, что отличия в них минимальны и состоят в расширении:

- терминального алфавита грамматики ЯВУ PL/1 путем введения в него новых терминальных символов «*****» и «/»,
- перечня допустимых арифметических операций, применяемых в арифметическом выражении, находящемся в правой части оператора присваивания, операцией **умножения «*»** и операцией **деления «/»**.

Понятно, что отличия, обнаруженные в исходных текстах условного персонального задания и демо-примера, обусловят отличия их эквивалентов на языке Ассемблер. На рис.

3 приведен один из возможных вариантов такого эквивалента для условного персонального задания, приведенного на рис. 2.

Метка	КОП	Операнды	Пояснения
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 3

Для определения объема приращения функционала макетного варианта утилиты компоновки, эмуляции и отладки, исходно рассчитанного только для обработки эквивалента демо-примера (см. рис. 2), с целью обеспечения возможности компиляции эквивалента на Ассемблере для условного персонального задания (см. рис. 3) нужно сравнить эти тексты и выделить строки с «новым» синтаксисом. Сделав это, убедимся, что к таким строкам следует отнести строки ассемблеровского эквивалента персонального задания с операциями умножения (**M**) и деления (**D**), которые имеют следующий синтаксис:

Метка	КОП	Операнды	Пояснения
	M	RRAB1,A	
	D	RRAB1,B	

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту г)

Из трех функций утилиты: компоновка, эмуляция и отладка для адаптации к «новым» машинным командам **умножения** и **деления** точно потребуется модификация только одного эмулятора, который является исполнителем «новых» машинных команд и, возможно, отладчика, который визуализирует изображение машинных команд на экране. Компоновщик, реализованный как абсолютный загрузчик, модификации не подлежит, т.к. условный персональный вариант структурно подобен демо-примеру, точнее, оба имеют по одной секции и компонуются в исполняемый код одинаково.
Изучая исходный текст утилиты, опираясь на построчные комментарии видим, что, как эмулятор, так и отладчик в своей работе используют общий элемент **БД** таблицу машинных операций **TMOP**, в которой исходно отсутствуют «новые» машинные команды. Следовательно, **TMOP** нужно модифицировать вводом двух дополнительных строк для машинных команд умножения и деления.
Т.к. «новые» машинные команды имеют тип **RX,** а обработка особенности типа, как в эмуляторе, так и в отладчике уже заложена в соответствующие функциональный блок **FRX**, то дополнительной модификации блока **FRX** не потребуется.
Алгоритм эмулятора предусматривает реализацию семантики каждой машинной команды в отдельной С-подпрограмме, поэтому для «новых» машинных команд умножения **(M)** и деления **(D)** потребуется составить две «новые» семантические С-подпрограммы:

```
int P_M(void) {
. . .
}
```

и

```
int P_D(void) {
. . .
}
```

Для обеспечения возможности активации этих С-подпрограмм в процессе эмуляции необходимо модифицировать подпрограмму **int sys(void)**, эмулирующую работу АЛУ реального процессора IBM/370:

```
int sys(void) {
. . .
SKIP:
Switch (T_MOP[k].CODOP) {
. . .
case '\x5C' : P_M();
              break;
. . .
case '\x5D' : P_D();
              break;

. . .
}
```

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту d)

Модифицировать утилиту, согласно перечню изменений из предыдущего пункта рекомендаций, следует на своем рабочем месте, опираясь на личный опыт работы в linux-среде.

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту е)

Отлаживать модифицированную в предыдущем пункте рекомендаций утилиту, следует на своем рабочем месте, опираясь на личный опыт работы в linux-среде и используя на входе объектный эквивалент условного персонального задания.

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту ж)

Отладку модифицированной утилиты для: загрузки, эмуляции и отладки следует в нашем случае выполнять, подавая на вход объектный эквивалент исходного кода условного персонального задания, до тех пор пока в по адресам размещения в ОЗУ прикладных переменных **D** и **E**, согласно алгоритму условного персонального задания, не появятся в конце прогона значения **2** и **1**, соответственно.
После завершения отладки утилиты следует продемонстрировать ее работоспособность Заказчику, т.е. преподавателю.

Если появились проблемы и/или вопросы, то идем с ними за консультацией к преподавателю.

Действия по пункту з)

Отчет по результатам составляется с учетом «**Требований к содержанию отчета**», приведенных, как в разделе 4 **Пособия**, так и ниже:
« **. . .**

Отчет должен соответствовать рекомендациям по оформлению студенческих курсовых работ, размещенных на сайте университета, и в общем случае должен содержать следующие разделы:

- описания примера из личного задания (указать формулировку личного задания),
- постановки задачи модификации программы-макета (описание выбранного способа модификации программы-макета),
- содержащие перечень и пояснения модификаций состава и (или) структуры БД утилиты,

- перечень и пояснения модификаций алгоритма утилиты (возможные формы описаний в виде: блок-схемы, текста на Си с комментариями, а в очевидных случаях обычным текстом),
- результат экспериментальной проверки работоспособности модифицированной утииты,
- выводы по результатам работы (указать позитивные и негативные моменты, проявившиеся в процессе работы, с общей личной оценкой проделанной работы).

. . . »

Если появились проблемы и/или вопросы, то идем с ними за как в консультацией к преподавателю.

Литература

1. Расторгуев В.Я. Разработка элементов учебной системы программирования. Абсолютный загрузчик [электронный ресурс: учебное пособие], СПбПУ, Спб., 2012

Как оформить отчет по результатам разработки утилиты загрузки, эмуляции и отладки исполняемого кода ?

В качестве аналога можно взять пример отчета для компилятора с ЯВУ (см. Рекомендация_4), где приведена структура полного отчета по КР, воспроизведенная ниже (см. рис. 1), и далее выполнить действия, связанные с построением части структуры, относящейся к отчету по утилите загрузки, эмуляции и отладке (далее просто **утилите**). Ниже приведена детализация этих действий с привязкой к разделам этого отчета по **утилите**.

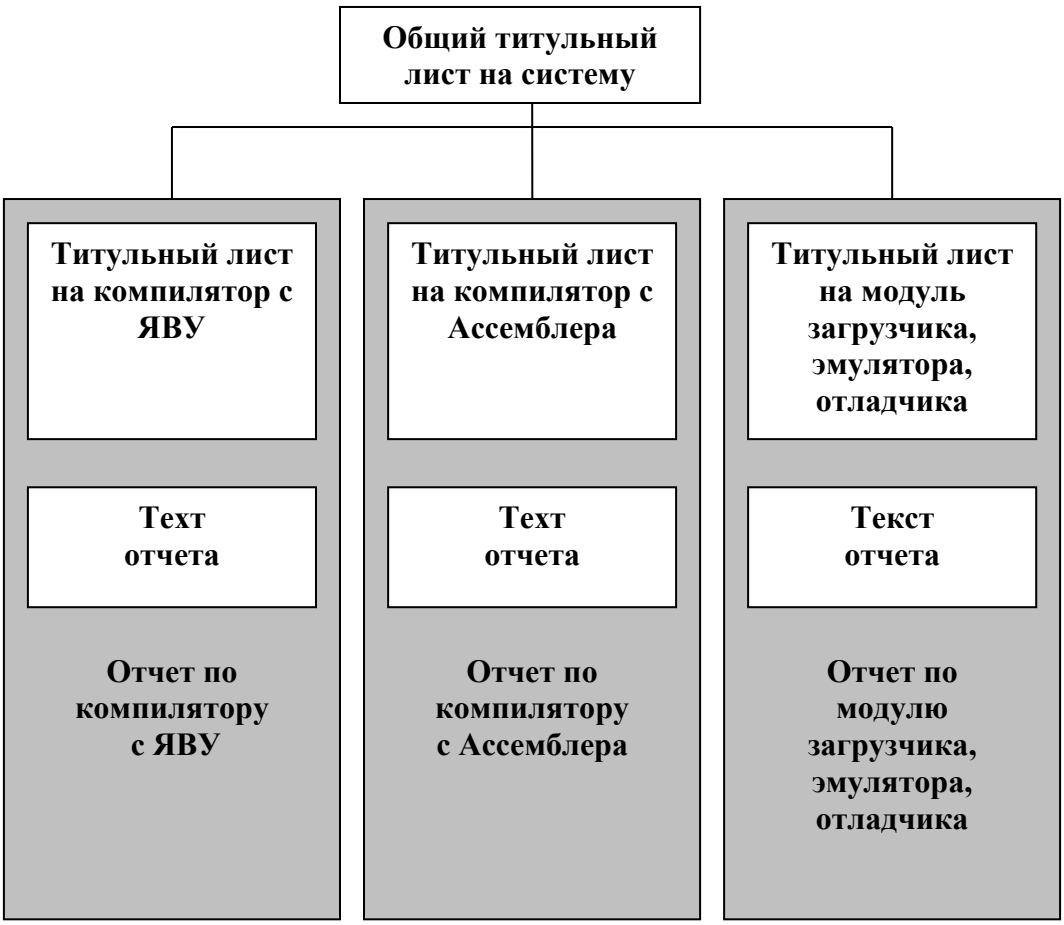

Рис. 1.

Титульный лист на элемент системы

Так как под элементом системы программирования здесь понимается **утилита**, то частный титульный лист для этой части КР следует оформлять, согласно Приложению1.

Согласно разделу 4 «**Требований к содержанию отчета**» из методического пособия [1], в материал отчета следует включить:
« . . .
Отчет должен соответствовать рекомендациям по оформлению студенческих курсовых работ, размещенных на сайте университета, и в общем случае должен содержать следующие разделы:

- Описания примера из личного задания (указать формулировку личного задания),
- Постановки задачи модификации программы-макета (описание выбранного способа модификации программы-макета),
- Перечень и пояснения модификаций состава и (или) структуры БД утилиты,
- Перечень и пояснения модификаций алгоритма утилиты (возможные формы описаний в виде: блок-схемы, текста на Си с комментариями, а в очевидных случаях обычным текстом),
- Результаты экспериментальной проверки работоспособности модифицированного макета утилиты.
- Выводы по результатам работы (указать позитивные и негативные моменты, проявившиеся в процессе работы, с общей личной оценкой проделанной работы).

. . .»

Далее будут приведены примеры выполнения требований к разделам отчета, основанные на условном персональном варианте задания, взятом из Рекомендаций_3, точнее говоря, на ассемблеровском эквиваленте условного персонального задания, который воспроизведен ниже (см. рис. 2):

Эквивалент условного персонального задания на Ассемблере

Метка	КОП	Операнды	Пояснения
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5

RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 2

и на объектном эквиваленте условного персонального задания, который взят из Рекомендаций_5 и воспроизведен ниже на рис. 3:

**********************   **Начало пакета объектных карт**   ***********************

	Тип карты		Имя	Тип им.	Отн. адрес	Длина	
	'ESD'		**EXAMP1**	**0**	**0x00**	**0x38**	

	Тип карты		адр		дл		Фрагмент двоичного кода	
	'TXT'		**0x00**		**2**		**05 50**	

	Тип карты		адр		дл		Фрагмент двоичного кода	
	'TXT'		**0x02**		**4**		**58 30 50 22**	

	Тип карты		адр		дл		Фрагмент двоичного кода	
	'TXT'		**0x06**		**4**		**5A 30 50 26**	

	Тип карты		адр		дл		Фрагмент двоичного кода	
	'TXT'		**0x0A**		**4**		**5B 30 50 2A**	

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x12**	**4**	**58 70 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x16**	**4**	**5C 60 50 22**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1A**	**4**	**5D 60 50 24**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1E**	**4**	**50 70 50 32**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x22**	**2**	**07FE**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x24**	**4**	**00 00 00 03**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x28**	**4**	**00 00 00 04**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x2C**	**4**	**00 00 00 05**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x30**	**4**	**XX XX XX XX**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x34**	**4**	**XX XX XX XX**

Тип карты	
‘END’	

********** **Конец пакета объектных карт** **********

Рис. 3

Пояснения к разделу «Задание на КР»

В этом разделе отчета логично дать формулировку всего задания на КР, уточнить свой персональный вариант и далее сфокусировать внимание на тех его аспектах, результаты работы над которыми войдут в настоящий отчет.
Пример реализации раздела отчета приведен в приложении 2.

Пояснения к разделу «Постановка задачи модификации макета в части утилиты загрузки, эмуляции и отладки»

В этом разделе следует привести постановку задачи (техническое задание) на ту часть КР, которая документируется данным отчетом. В данном случае это утилита загрузки, эмуляции и отладки, которая должна быть получена путем модификации функционала своего макетного прототипа под персональный вариант задания.
Понятно, что, согласно существующей практике, черновик ТЗ составляет исполнитель (студент), после чего согласует его с преподавателем, далее приступает к его реализации и переносит его в отчет. В ТЗ должны быть определены:

- предмет разработки,
- требования по входу и выходу,
- опционно, при согласовании могут быть предложены Заказчику (преподавателю) и согласованы с ним мотивированные ограничения функционала предмета разработки.

Пример реализации раздела отчета приведен в приложении 3.

Пояснения к разделу «Модификация состава и структуры БД макета утилиты загрузки, эмуляции и отладки»

В этом разделе отчета следует указать модификации элементов БД для каждого функционального блока **утилиты**, т.е. для загрузчика (компоновщика), эмулятора и отладчика.

Пример реализации раздела отчета приведен в приложении 4.

Пояснения к разделу «Модификация алгоритма макета утилиты загрузки, эмуляции и отладки»

В этом разделе отчета следует указать подпрограммы, нуждающиеся в модификациях, для каждого функционального блока **утилиты**, т.е. для загрузчика (компоновщика), эмулятора и отладчика.
Пример реализации раздела отчета приведен в приложении 5.

Пояснения к разделу «Результаты экспериментальной проверки модифицированной утилиты закрузки, эмуляции и отладки»

После отладки модифицированного макета до состояния, когда в момент завершения работы утилиты, т.е. после завершения отладки персонального варианта сработает критерий подтверждения работоспособности, сформулированный в разделе «Постановка задачи», пишем этот раздел отчета.
Пример реализации этого раздела приведен в приложении 6.

Пояснения к разделу «Выводы по результатам работы»

В начале раздела необходимо разместить две обязательные фразы.

Первая фраза отражает самооценку разработчика в одном из двух вариантов:

1) Порученное задание мною выполнено успешно.
2) Порученное задание мною не выполнено.

Вторая фраза доказательно подтверждает первую фразу в одном из двух возможных вариантов:

1) Успешность выполнения порученного задания подтверждена выполнением критерия подтверждения работоспособности, согласованного при постановке задачи, полученного при экспериментальной проверке утилиты загрузки, эмуляции и отладки.
2) Неудача выполнения порученного задания подтверждена невыполнением критерия подтверждения работоспособности, согласованного при постановке задачи, полученного при экспериментальной проверке утилиты загрузки, эмуляции и отладки.
3) .

Третья и последующие фразы добавляются исполнителем опционно и без особой надежды на то, что эти фразы будут внимательно прочитаны Заказчиком.

Пример реализации этого раздела приведен в приложении 7.

Пояснения ко всему отчету в целом

Пример целого отчета, в котором собраны все формулировки разделов, рекомендованные выше, приведен в приложении 8.

Литература

1. Расторгуев В.Я. Разработка учебной системы программирования. Абсолютный загрузчик. [Электронный ресурс : учебное пособие], СПбПУ, СПб, 2012

Приложение 1

Санкт-Петербургский политехнический университет
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Разработка учебной системы программирования
Утилита загрузки, эмуляции и отладки

Дисциплина: «Системы программирования»
Вариант № XX

Исполнители:

Студенты группы ……. Иванов И.И.

Петров П.П.

Руководитель Расторгуев В.Я.

Санкт-Петербург

2020

Приложение 2

1. Задание на курсовую работу

Задание на КР дано в следующей общей формулировке:

«Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте».

Общая формулировка персонализирована в варианте № XX, приведенном на рис. 1.

Условный персональный вариант № XX

```
EXAMP1: PROC OPTIONS ( MAIN );
        DCL A BIN FIXED ( 31 ) INIT ( 111B );
        DCL B BIN FIXED ( 31 ) INIT ( 100B );
        DCL C BIN FIXED ( 31 ) INIT ( 101B );
        DCL D BIN FIXED ( 31 );
        DCL E BIN FIXED ( 31 );
        D = A + B - C;
        E = D * A / B;
END EXAMP1;
```

Рис. 1

Схема функционирования макета учебной системы программирования, являющегося, согласно заданию на КР, одним из элементов исходных данных, приведена на рис. 2.

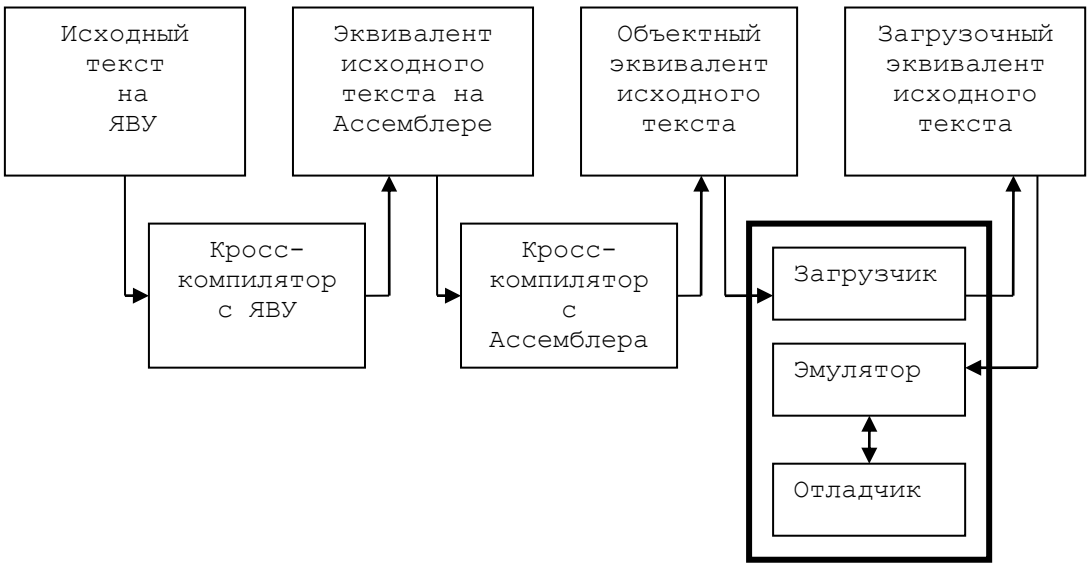

Рис. 2

Дополнительные требования к КР, вытекающие из рис.2, предполагают то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (IBM 370). 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 IBM 370, 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 IBM 370 и хранится в виде двоичного файла технологической ЭВМ с расширением *.tex;
- загрузочный эквивалент исходной программы представляет собой машинный код IBM 370, запоминаемый в области ОЗУ технологической ЭВМ, являющейся зоной загрузки для эмулятора объектной ЭВМ.

На рис. 2 жирным контуром выделена утилита загрузки, эмуляции и отладки исполняемого кода персонального варианта задания, являющаяся тем элементом макета, результаты модификации функционала которого под персональный вариант № XX (см. рис 1) приведены в данном отчете.

Согласно проектным решениям, реализованным при разработке компилятора с ЯВУ, после компиляции условного персонального задания (см. рис. 1) должен быть получен следующий его эквивалент на Ассемблере (см. рис. 3):

Метка	**КОП**	**Операнды**	**Пояснения**
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E

RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14
RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 3

Согласно рис. 2, этот текст на Ассемблере должен быть входом для компилятора с Ассемблера, который, в свою очередь, должен на выходе формировать из него (см. рис. 4) объектный эквивалент следующего содержания,

********** **Начало пакета объектных карт** **********

	Тип карты		Имя	Тип им.	Отн. адрес	Длина	
	‘ESD’		**EXAMP1**	**0**	**0x00**	**0x38**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x00**		**2**		**05 50**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x02**		**4**		**58 30 50 22**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x06**		**4**		**5A 30 50 26**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x0A**		**4**		**5B 30 50 2A**	

Тип карты	**адр**	**дл**	**Фрагмент двоичного кода**
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	**адр**	**дл**	**Фрагмент двоичного кода**
‘TXT’	**0x12**	**4**	**58 70 50 2E**

Тип карты	**адр**	**дл**	**Фрагмент двоичного кода**
‘TXT’	**0x16**	**4**	**5C 60 50 22**

Тип карты	**адр**	**дл**	**Фрагмент двоичного кода**
‘TXT’	**0x1A**	**4**	**5D 60 50 24**

Тип карты	**адр**	**дл**	**Фрагмент двоичного кода**
‘TXT’	**0x1E**	**4**	**50 70 50 32**

Тип карты	**адр**	**дл**	**Фрагмент двоичного кода**
‘TXT’	**0x22**	**2**	**07FE**

Тип карты	**адр**	**дл**	**Фрагмент двоичного кода**
‘TXT’	**0x24**	**4**	**00 00 00 03**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x28**	**4**	**00 00 00 04**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x2C**	**4**	**00 00 00 05**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x30**	**4**	**XX XX XX XX**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x34**	**4**	**XX XX XX XX**

Тип карты	
‘END’	

********** **Конец пакета объектных карт** **********

Рис. 4.

который является входом для утилиты загрузки, эмуляции и отладки.

Приложение 3

2. Постановка задачи модификации макета в части утилиты загрузки, эмуляции и отладки

Схема функционирования модифицированной макетной утилиты загрузки, эмуляции и отладки должна соответствовать рис. 5.

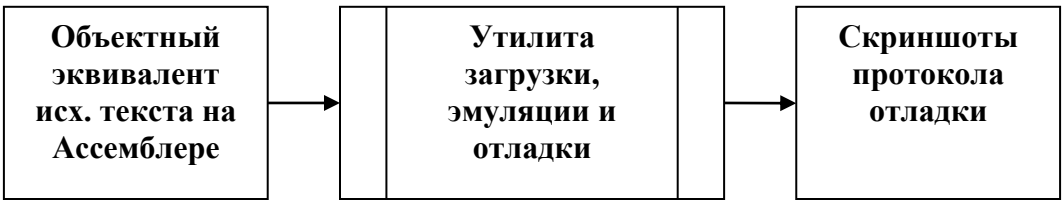

Рис. 5

Функционал модифицированной макетной утилиты должен для объектного эквивалента (см. рис 4) обеспечить:

- Формирование **компоновщиком** исполняемого кода программы персонального варианта задания и загрузки его ОЗУ эмулятора,
- Покомандное выполнение исполняемого кода **на эмуляторе**,
- Управление **из отладчика** покомандным режимом работы эмулятора с выдачей протокола отладки.

За **критерий подтверждения правильного функционирования модифицированной утилиты** может быть принят факт появления по адресам размещения финишных прикладных переменных **D** и **E**, соответственно, значений **2** и **1**. Это следует из алгоритма условного персонального задания.
Учитывая линейность исполняемой программы для проверки срабатывания этого критерия в отчете достаточно представить из всего протокола отладки только двух скриншотов, а именно:

- Начального скриншота, который подтверждает пустые стартовые значения переменных **D** и **E**,
- Конечного скриншота, который подтвердит заявленные в критерии финишные значения переменных **D** и **E**.

Из трех перечисленных функциональных блоков утилиты: компоновщика, эмулятора и отладчика модификацию в интересах персонального варианта задания следует делать только для эмулятора.

Функционал макетного компоновщика достаточен для односекционных проектов приложений, к числу которых относится и условный персональный вариант. Поэтому, компоновщик в модификации не нуждается.

Функционал макетного отладчика достаточен для визуализации «новых» машинных команд **деления (D)** и **умножения (M)**. Поэтому, отладчик в модификации не нуждается.

Согласованные функциональные ограничения

По согласованию с преподавателем в модифицированной макетной утилите загрузки, эмуляции и отладки введены следующие функциональные ограничения:

1) . . .
2) . . .

3. Модификация состава и структуры БД макета утилиты загрузки, эмуляции и отладки

Выше было показано, что для адаптации функционала макета **утилиты** под персональный вариант задания необходимо модифицировать только эмулятор, который в своей работе опирается на таблицу машинных операций **TMOP**, которая настроена на демо-пример и имеет следующий вид:

Мнемоника	**КОП**	**Длина в байтах**	**Т**ип
BALR	0x05	2	RR
L	0x58	4	RX
A	0x5A	4	RX
S	0x5B	4	RX
ST	0x50	4	RX
BCR	0x07	2	RR

Рис. 6. TMOP – таблица машинных операций (команд) для демо-примера

Для условного персонального задания необходима возможность использования новых машинных команд: **умножения (M)** и **деления (D)**. Для этого необходимо модифицировать таблицу допустимых машинных команд **TMOP** (см. рис. 6), добавив две новые строки для команд **M** и **D** (см. рис. 7), которые на рис. 7 выделены жирным шрифтом.

Мнемоника	**КОП**	**Длина в байтах**	**Т**ип
BALR	0x05	2	RR
L	0x58	4	RX
A	0x5A	4	RX
S	0x5B	4	RX
ST	0x50	4	RX
M	**0x5C**	**4**	**RX**
D	**0x5D**	**4**	**RX**
BCR	0x07	2	RR

Рис. 7. TMOP – таблица машинных операций (команд) для условного персонального задания

4. Модификация алгоритма утилиты загрузки, эмуляции и отладки

Выше было показано, что для адаптации функционала макета под персональный вариант задания необходимо модифицировать только макетный эмулятор в части настройки его на возможность выполнения «новых» машинных команд **умножения (M)** и **деления (D)**. Алгоритм эмулятора предусматривает реализацию семантики каждой машинной команды в отдельной С-подпрограмме, поэтому для «новых» машинных команд умножения **(M)** и деления **(D)** потребовалось составить две «новые» семантические С-подпрограммы:

```
int P_M(void) {
. . .
/* Здесь следует расположить код на языке Си, реализующий семантику машинной
команды умножения */
. . .
}
```

и

```
int P_D(void) {
. . .
/* Здесь следует расположить код на языке Си, реализующий семантику машинной
команды деления */
. . .
}
```

Для обеспечения возможности активации этих С-подпрограмм в процессе эмуляции персонального варианта задания была модифицирована подпрограмма **int sys(void)**, эмулирующая работу АЛУ реального процессора IBM/370:

```
int sys(void) {
. . .
SKIP:
Switch (T_MOP[k].CODOP) {
. . .
case '\x5C' : P_M();        /* активация машинной операции умножения в случае
              break;           ее обнаружения в последовательности исполняемых
. . .                          машинных команд приложения */

case '\x5D' : P_D();        /* активация машинной операции деления в случае
              break;           ее обнаружения в последовательности исполняемых
                               машинных команд приложения */
. . .
}
```

Приложение 6

5. Результаты экспериментальной проверки модифицированной утилиты загрузки, эмуляции и отладки

Работоспособность модифицированной под персональный вариант задания макета утилиты загрузки, эмуляции и отладки была подтверждена испытательным прогоном ее в среде ОС Linux. В результате был проведен сеанс отладки персонального варианта задания и получен протокол отладки. Стартовый и финишный скриншоты этого протокола приведены на рис. 8 и рис. 9.

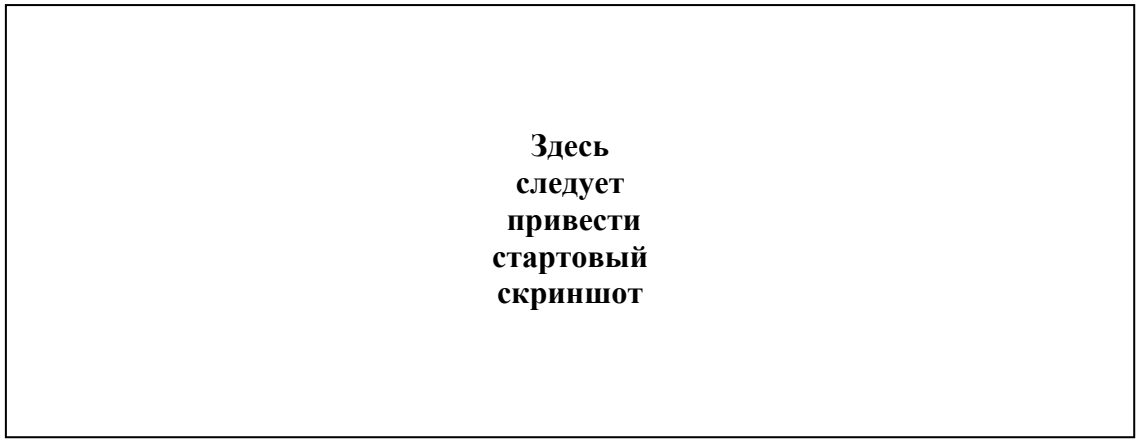

Рис. 8

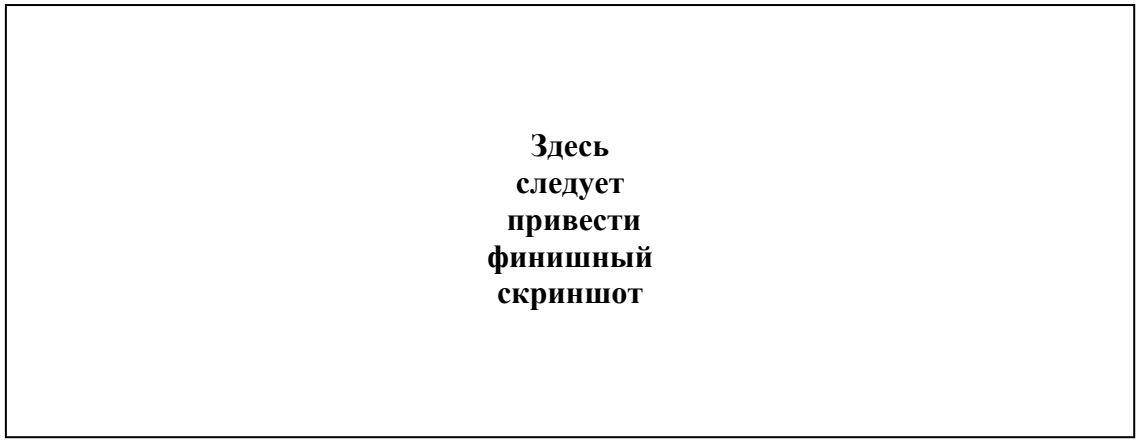

Рис. 9

Сравнивая стартовый и финишный скриншоты в адресах ОЗУ, отведенных для хранения переменных **D** (адрес XX XX XX) и **E** (адрес YY YY YY), легко видеть, что в конце отладки они приняли, соответственно, значения **2** и **1**, что удовлетворяет критерию подтверждения работоспособности, сформулированному в разделе «Постановка задачи».

Приложение 7

6. Выводы по результатам работы

Порученное задание выполнено успешно. Успешность выполнения порученного задания подтверждена выполнением критерия подтверждения работоспособности, согласованного при постановке задачи, полученного при экспериментальной проверке утилиты загрузки, эмуляции и отладки.

Приложение 8

Санкт-Петербургский политехнический университет
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я Р А Б О Т А

Разработка учебной системы программирования
Утилита загрузки, эмуляции и отладки

Дисциплина: «Системы программирования»
Вариант № XX

Исполнители:

Студенты группы ……. Иванов И.И.

Петров П.П.

Руководитель Расторгуев В.Я.

Санкт-Петербург

2020

2. Задание на курсовую работу

Задание на КР дано в следующей общей формулировке:

«Необходимо выполнить доработку элементов макета учебной системы программирования до уровня, позволяющего обрабатывать «новые» для макета конструкции языка высокого уровня, примененные в соответствующем варианте».

Общая формулировка персонализирована в варианте № XX, приведенном на рис. 1.

Условный персональный вариант № XX

```
EXAMP1: PROC OPTIONS ( MAIN );
        DCL A BIN FIXED ( 31 ) INIT ( 111B );
        DCL B BIN FIXED ( 31 ) INIT ( 100B );
        DCL C BIN FIXED ( 31 ) INIT ( 101B );
        DCL D BIN FIXED ( 31 );
        DCL E BIN FIXED ( 31 );
        D = A + B - C;
        E = D * A / B;
END EXAMP1;
```

Рис. 1

Схема функционирования макета учебной системы программирования, являющегося, согласно заданию на КР, одним из элементов исходных данных, приведена на рис. 2.

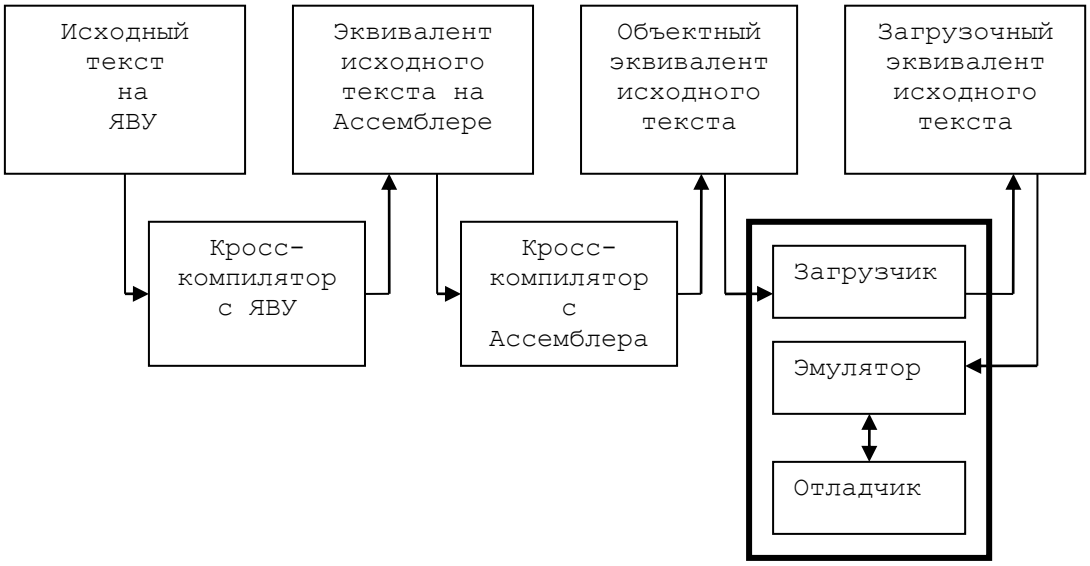

Рис. 2

Дополнительные требования к КР, вытекающие из рис.2, предполагают то, что данная система программирования работает на технологической ЭВМ (IBM PC) и является по существу кросс-системой для объектной ЭВМ (IBM 370). 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 IBM 370, 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 IBM 370 и хранится в виде двоичного файла технологической ЭВМ с расширением *.tex;
- загрузочный эквивалент исходной программы представляет собой машинный код IBM 370, запоминаемый в области ОЗУ технологической ЭВМ, являющейся зоной загрузки для эмулятора объектной ЭВМ.

На рис. 2 жирным контуром выделена утилита загрузки, эмуляции и отладки исполняемого кода персонального варианта задания, являющаяся тем элементом макета, результаты модификации функционала которого под персональный вариант № XX (см. рис 1) приведены в данном отчете.

Согласно проектным решениям, реализованным при разработке компилятора с ЯВУ, после компиляции условного персонального задания (см. рис. 1) должен быть получен следующий его эквивалент на Ассемблере (см. рис. 3):

Метка	**КОП**	**Операнды**	**Пояснения**
EXAMP1	START	0	счетчик отн. адреса в нуль
	BALR	RBASE,0	отн. адр. базы в RBASE
	USING	*,RBASE	Объявл. RBASE регистром базы
	L	RRAB,A	Загрузка A в регистр RRAB
	A	RRAB,B	A+B грузим в RRAB
	S	RRAB,C	A+B-C грузим в RRAB
	ST	RRAB,D	A+B-C из RRAB грузим в D
	L	RRAB1+1,D	Загрузка D в регистр RRAB1+1
	M	RRAB1,A	D*A грузим в RRAB1 и RRAB1+1
	D	RRAB1,B	D*A/B→RRAB1+1
	ST	RRAB1+1,E	D*A/B из RRAB1+1 грузим в E
	BCR	15,RVIX	Переход по адр. в рег. RVIX
A	DC	F’3’	A=3
B	DC	F’4’	B=4
C	DC	F’5’	C=5
D	DS	F	Резерв памяти для D
E	DS	F	Резерв памяти для E
RBASE	EQU	5	RBASE назначим 5
RRAB	EQU	3	RRAB назначим 3
RVIX	EQU	14	RVIX назначим 14

RRAB1	EQU	6	RRAB1 назначим 6
	END		Конец текста блока

Рис. 3

Согласно рис. 2, этот текст на Ассемблере должен быть входом для компилятора с Ассемблера, который, в свою очередь, должен на выходе формировать из него (см. рис. 4) объектный эквивалент следующего содержания,

********************** Начало пакета объектных карт *************************

	Тип карты		Имя	Тип им.	Отн. адрес	Длина	
	‘ESD’		**EXAMP1**	**0**	**0x00**	**0x38**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x00**		**2**		**05 50**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x02**		**4**		**58 30 50 22**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x06**		**4**		**5A 30 50 26**	

	Тип карты		**адр**		**дл**		**Фрагмент двоичного кода**	
	‘TXT’		**0x0A**		**4**		**5B 30 50 2A**	

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x0E**	**4**	**50 30 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x12**	**4**	**58 70 50 2E**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x16**	**4**	**5C 60 50 22**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1A**	**4**	**5D 60 50 24**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x1E**	**4**	**50 70 50 32**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x22**	**2**	**07FE**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x24**	**4**	**00 00 00 03**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x28**	**4**	**00 00 00 04**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x2C**	**4**	**00 00 00 05**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x30**	**4**	**XX XX XX XX**

Тип карты	адр	дл	Фрагмент двоичного кода
‘TXT’	**0x34**	**4**	**XX XX XX XX**

Тип карты	
‘END’	

********** **Конец пакета объектных карт** **********

Рис. 4.

который является входом для утилиты загрузки, эмуляции и отладки.

2. Постановка задачи модификации макета в части утилиты загрузки, эмуляции и отладки

Схема функционирования модифицированной макетной утилиты загрузки, эмуляции и отладки должна соответствовать рис. 5.

Рис. 5

Функционал модифицированной макетной утилиты должен для объектного эквивалента (см. рис 4) обеспечить:

- Формирование **компоновщиком** исполняемого кода программы персонального варианта задания и загрузки его ОЗУ эмулятора,
- Покомандное выполнение исполняемого кода **на эмуляторе**,
- Управление **из отладчика** покомандным режимом работы эмулятора с выдачей протокола отладки.

За **критерий подтверждения правильного функционирования модифицированной утилиты** может быть принят факт появления по адресам размещения финишных прикладных переменных **D** и **E**, соответственно, значений **2** и **1**. Это следует из алгоритма условного персонального задания.
Учитывая линейность исполняемой программы для проверки срабатывания этого критерия в отчете достаточно представить из всего протокола отладки только двух скриншотов, а именно:

- Начального скриншота, который подтверждает пустые стартовые значения переменных **D** и **E**,
- Конечного скриншота, который подтвердит заявленные в критерии финишные значения переменных **D** и **E**.

Из трех перечисленных функциональных блоков утилиты: компоновщика, эмулятора и отладчика модификацию в интересах персонального варианта задания следует делать только для эмулятора.

Функционал макетного компоновщика достаточен для односекционных проектов приложений, к числу которых относится и условный персональный вариант. Поэтому, компоновщик в модификации не нуждается.

Функционал макетного отладчика достаточен для визуализации «новых» машинных команд **деления (D)** и **умножения (M)**. Поэтому, отладчик в модификации не нуждается.

Согласованные функциональные ограничения

По согласованию с преподавателем в модифицированной макетной утилите загрузки, эмуляции и отладки введены следующие функциональные ограничения:

3) . . .
4) . . .

3. Модификация состава и структуры БД макета утилиты загрузки, эмуляции и отладки

Выше было показано, что для адаптации функционала макета **утилиты** под персональный вариант задания необходимо модифицировать только эмулятор, который в своей работе опирается на таблицу машинных операций **TMOP**, которая настроена на демо-пример и имеет следующий вид:

Мнемоника	**КОП**	**Длина в байтах**	**Т**ип
BALR	0x05	2	RR
L	0x58	4	RX
A	0x5A	4	RX
S	0x5B	4	RX
ST	0x50	4	RX
BCR	0x07	2	RR

Рис. 6. TMOP – таблица машинных операций (команд) для демо-примера

Для условного персонального задания необходима возможность использования новых машинных команд: **умножения (M)** и **деления (D)**. Для этого необходимо модифицировать таблицу допустимых машинных команд **TMOP** (см. рис. 6), добавив две новые строки для команд **M** и **D** (см. рис. 7), которые на рис. 7 выделены жирным шрифтом.

Мнемоника	**КОП**	**Длина в байтах**	**Т**ип
BALR	0x05	2	RR
L	0x58	4	RX
A	0x5A	4	RX
S	0x5B	4	RX
ST	0x50	4	RX
M	**0x5C**	**4**	**RX**
D	**0x5D**	**4**	**RX**
BCR	0x07	2	RR

Рис. 7. TMOP – таблица машинных операций (команд) для условного персонального задания

4. Модификация алгоритма утилиты загрузки, эмуляции и отладки

Выше было показано, что для адаптации функционала макета под персональный вариант задания необходимо модифицировать только макетный эмулятор в части настройки его на возможность выполнения «новых» машинных команд **умножения (M)** и **деления (D)**.

Алгоритм эмулятора предусматривает реализацию семантики каждой машинной команды в отдельной С-подпрограмме, поэтому для «новых» машинных команд умножения **(M)** и деления **(D)** потребовалось составить две «новые» семантические С-подпрограммы:

```
int P_M(void) {
. . .
/* Здесь следует расположить код на языке Си, реализующий семантику машинной
команды умножения */
. . .
}
```

и

```
int P_D(void) {
. . .
/* Здесь следует расположить код на языке Си, реализующий семантику машинной
команды деления */
. . .
}
```

Для обеспечения возможности активации этих С-подпрограмм в процессе эмуляции персонального варианта задания была модифицирована подпрограмма **int sys(void)**, эмулирующая работу АЛУ реального процессора IBM/370:

```
int sys(void) {
. . .
SKIP:
Switch (T_MOP[k].CODOP) {
. . .
case '\x5C' : P_M();        /* активация машинной операции умножения в случае
              break;           ее обнаружения в последовательности исполняемых
. . .                          машинных команд приложения */

case '\x5D' : P_D();        /* активация машинной операции деления в случае
              break;           ее обнаружения в последовательности исполняемых
                               машинных команд приложения */
. . .
}
```

5. Результаты экспериментальной проверки модифицированной утилиты загрузки, эмуляции и отладки

Работоспособность модифицированной под персональный вариант задания макета утилиты загрузки, эмуляции и отладки была подтверждена испытательным прогоном ее в среде ОС Linux. В результате был проведен сеанс отладки персонального варианта задания и получен протокол отладки. Стартовый и финишный скриншоты этого протокола приведены на рис. 8 и рис. 9.

Рис. 8

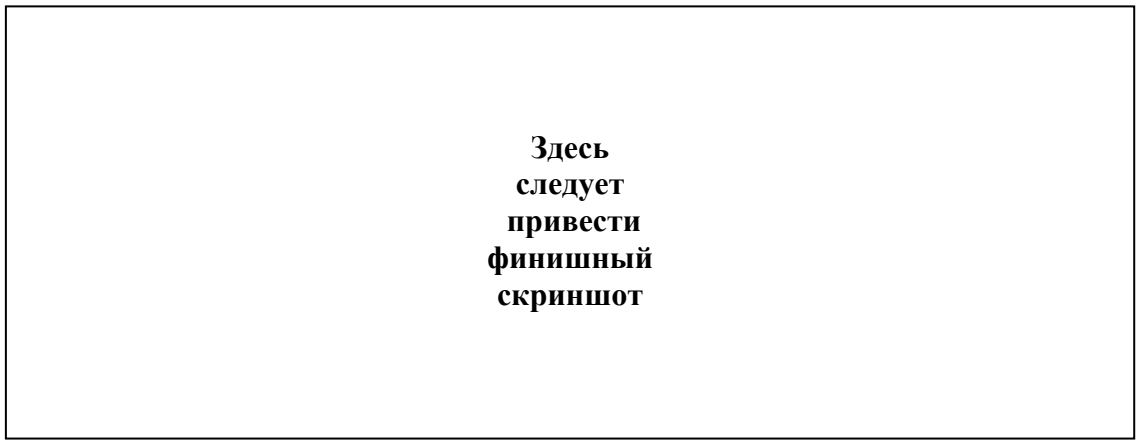

Рис. 9

Сравнивая стартовый и финишный скриншоты в адресах ОЗУ, отведенных для хранения переменных **D** (адрес XX XX XX) и **E** (адрес YY YY YY), легко видеть, что в конце отладки они приняли, соответственно, значения **2** и **1**, что удовлетворяет критерию подтверждения работоспособности, сформулированному в разделе «Постановка задачи».

6. Выводы по результатам работы

Порученное задание выполнено успешно. Успешность выполнения порученного задания подтверждена выполнением критерия подтверждения работоспособности, согласованного при постановке задачи, полученного при экспериментальной проверке утилиты загрузки, эмуляции и отладки.