Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
«Севастопольский государственный университет»

Институт информационных технологий и управления в технических системах

М Е Т О Д И Ч Е С К И Е У К А З А Н И Я

к выполнению курсовой работы
«Разработка системы программирования
для языка Ассемблер»
п о д и с ц и п л и н е
«Системное программирование»
для студентов очной и заочной форм обучения
направления 09.03.01–«Информатика и вычислительная техника»

Севастополь

2016

УДК 004.451

Методические указания к выполнению курсовой работы «Разработка системы программирования для языка Ассемблер» по дисциплине «Системное программирование» для студентов очной и заочной форм обучения направления 09.03.01–«Информатика и вычислительная техника»/ Тертычный А.И., Шалимова Е.М. – Севастополь: Издательство СевГУ, 2016.- 20с.

Целью методических указаний к выполнению курсовой работы является изложение основных теоретических положений разработки системного программного обеспечения. В методических указаниях представлены требования к транслятору с языка Ассемблер и универсальному отладчику–эмулятору. Кроме того, в методических указаниях приведены рекомендации по разработке алгоритмов и выбору структур данных, порядок выполнения курсовой работы, варианты заданий и содержание пояснительной записки.

Методические указания утверждены на заседании кафедры ИТиКС, протокол № 1 от 1 сентября 2016 г.

Рецензент Апраксин Ю.К., д.т.н, профессор.

Ответственный за выпуск
заведующий кафедрой ИТиКС Брюховецкий А.А.

СОДЕРЖАНИЕ

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Программное обеспечение персональных ЭВМ (ПО) можно разделить на прикладное и специальное или системное. Прикладное ПО – это программы пользователей, системное – это операционные системы, компиляторы, трансляторы, отладчики, драйверы и т.д.

В рамках данной курсовой работы предполагается разработка системы программирования низкого уровня, включающей транслятор для подмножества команд языка Ассемблера IBM PC и универсальный отладчик-эмулятор.

Любая программа, переводящая программу на исходном языке в выходную или объектную программу, называется транслятором. Если исходным языком является Ассемблер, то транслятор тоже часто называют ассемблером.

Язык Ассемблер является машинно-ориентированным языком, другими словами, для разработки ассемблерных программ от программиста требуется знание организации всей системы компьютера: архитектуры, памяти, адресации и т.д. [1-7]. Абстрактная модель ЭВМ представлена на рисунке 1.

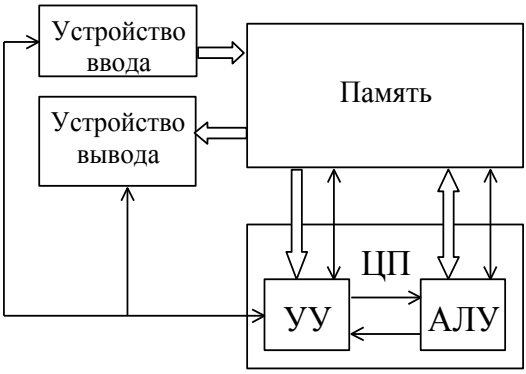


Рисунок 1– Абстрактная модель ЭВМ.

На рисунке 1 приняты следующие обозначения:

ЦП – центральный процессор;
УУ – устройство управления;
АЛУ – арифметико-логическое устройство;
⇒ передача данных;
→ передача управляющих сигналов.

Процессоры *i8086/8088* имеют 14 регистров, каждый из которых имеет длину в одно слово. Длина машинного слова – 2 байта или 16 бит. Всем регистрам присвоены имена, которые и используются для обращения к регистрам. На рисунке 2 приведена классификация регистров.

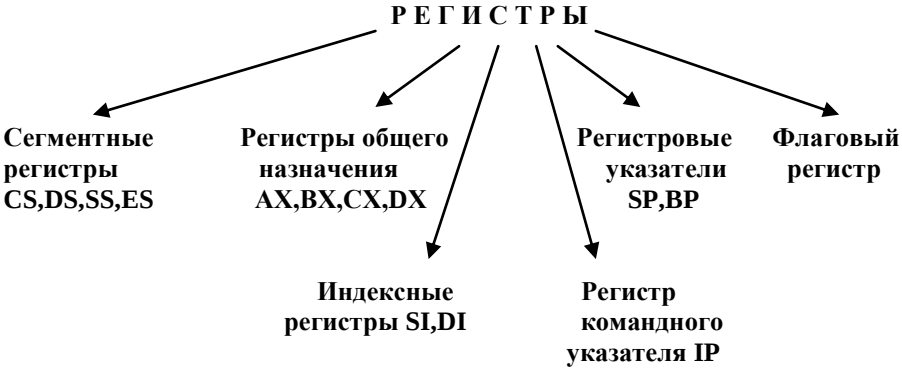


Рисунок 2– Классификация регистров процессора *i8086/8088*.

Регистры общего назначения состоят из двух 1-байтовых регистров, к которым можно обращаться независимо: AH, BH, CH, DH (старшие байты) и AL, BL, CL, DL (младшие байты).

Память компьютера представляется в виде линейной пронумерованной последовательности байт. Каждый байт имеет адрес (номер), т.е. байт – это минимальная адресуемая единица информации. Процессор может обращаться как к байтам, так и к словам памяти. Слово содержит два смежных байта памяти и может начинаться как с четного, так и нечетного адреса. При размещении слова в памяти байт с адресом, соответствующим адресу слова, содержит его младшую часть, а следующий байт содержит его старшую часть.

Логически память разбивается на сегменты размером по 64 Кбайт. Физический адрес памяти представляется 20-битовым числом и получается из двух 16-битных частей: адреса начала сегмента и смещения относительно начала сегмента. Физический адрес любого байта памяти формируется сложением следующих операндов:

0000	**Смещение**

+

Значение сегментного регистра	**0000**

При таком способе формирования адреса сегмент всегда начинается с адреса, кратного 16. Все пространство памяти делится на параграфы – области из 16 смежных байт, начиная с нулевого адреса. Очевидно, что

любой сегмент может начинаться только на границе параграфа (четыре младших бита адреса – нулевые).

В общем случае длина машинной команды от 1 до 6 байт. Код команды занимает первый байт, способ адресации задается во втором байте, в остальных байтах команды указываются непосредственные данные или адрес памяти, где эти данные находятся.

Структура адресного байта представлена на рисунке 3.

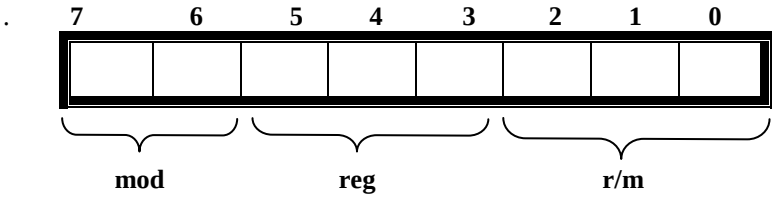


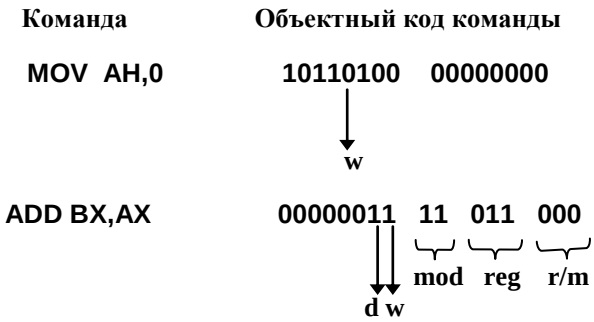
Рисунок 3 – Структура адресного байта

Байт кода команды содержит бит **w**, определяющий длину операндов команды:

w=0 – размер операндов – байт;

w=1 – размер операндов – слово.

Однако, положение бита **w** для разных команд может быть различно. Например,



Кроме того, первый байт машинного кода может содержать бит **d**, который указывает направление потока данных между операндами. Если **d=0**, то поле **reg** определяет второй операнд, а поля **mod** и **r/m** – первый, иначе поле **reg** определяет первый операнд, а поля **mod** и **r/m** – второй.

В последнем примере **d=1** означает, что поле **reg** определяет первый операнд, а поля **mod** и **r/m** - второй.

Операнды команды кодируется полями **reg** или **mod,r/m**. Поле **reg** предназначено для указания операнда с регистровым режимом адресации и содержит код регистра. Поле **r/m** предназначено для указания операнда с

регистровым режимом адресации или с режимом адресации оперативной памяти. Если биты поля **mod** имеют значение **11**, это значит, что операнд находится в регистре, т. е. биты **r/m** вместе с битом **w** определяют конкретный регистр. Кодирование регистров приведено в таблице 1. Если биты поля **mod** имеют значение **00,01,10** это значит, что операнд находится в оперативной памяти, т. е. поле **r/m** содержит код способа адресации оперативной памяти.

Таблица 1– Таблица соответствия кодов и мнемонических имен регистров

reg или **r/m**		**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
Регистр	**w =0**	**AL**	**CL**	**DL**	**BL**	**AH**	**CH**	**DH**	**BH**
	w=1	**AX**	**CX**	**DX**	**BX**	**SP**	**BP**	**SI**	**DI**

Значения битов **mod** имеют следующую интерпретацию:

mod=00 – биты **r/m** задают абсолютный адрес, байт смещения отсутствует;

mod=01 – биты **r/m** задают абсолютный адрес памяти и имеется один байт смещения;

mod=10 – биты **r/m** задают абсолютный адрес памяти и имеется два байта смещения.

Кодирование типов адресаций приведено в таблице 2.

Таблица 2– Таблица кодов адресации

r/m	**mod=00**	**mod=01 или mod=10**
000	**[BX]+[SI]**	**[BX]+[SI]+смещение**
001	**[BX]+[DI]**	**[BX]+[DI]+смещение**
010	**[BP]+[SI]**	**[BP]+[SI]+смещение**
011	**[BP]+[DI]**	**[BP]+[DI]+смещение**
100	**[SI]**	**[SI]+смещение**
101	**[DI]**	**[DI]+смещение**
110	**Прямая**	**[BP]+смещение**
111	**[BX]**	**[BX]+смещение**

В объектном коде команды, если используются операнды длиной в слово, сначала указывается младший байт, а затем – старший.

Например,

Команда	**Объектный код команды**
MOV AX, 0123H	**10111000 00100011 00000001**
младший байт **старший байт**	**младший байт** **старший байт**

В командах с одним операндом или в командах, когда второй операнд является непосредственным значением, первый операнд кодируется полями **mod** и **r/m**, а поле **reg** содержит дополнительные 3 бита кода операции.

Например,

Команда	**Объектный код команды**
MUL DH	**11111100 11 101 110**

1.1. ТРЕБОВАНИЯ К ТРАНСЛЯТОРУ

Транслятор с языка Ассемблер должен:

- выявить ошибки в исходной программе;
- распределить память;
- перевести на машинный язык команды мнемокода и константы;
- сформировать объектный код программы;
- сформировать протокол трансляции (листинг).

Эти задачи трудно решить за один просмотр [3–7]. Поэтому, как правило, используются двухпросмотровые ассемблеры. При первом просмотре ассемблер просматривает исходную программу и строит таблицу идентификаторов, т.е. имен полей данных и меток, используемых в программе, и их относительных адресов.

Кроме того, при первом проходе подсчитывается длина объектного кода, но сам объектный код не генерируется. При втором просмотре, ассемблер, используя таблицу идентификаторов, генерирует объектный код для каждой команды.

В ходе трансляции ассемблер использует постоянные и временные таблицы. Постоянные таблицы составляются при разработке ассемблера, а временные создаются в ходе трансляции. Примером постоянной таблицы может служить таблица команд, в которой перечислены все мнемонические коды команд, типы операндов и их машинные эквиваленты, или таблица сообщений об ошибках, в которой фиксируется тип ошибки и соответствующий текст для вывода в файл-листинг. Множество и структура таблиц зависят от особенностей машины, языка Ассемблер и, наконец, от конкретной реализации транслятора. Однако, обязательной временной таблицей в любом ассемблере является таблица идентификаторов (имен), содержащая все имена, определенные в исходной программе и их характеристики: тип, значение, длину и т.д.

Результатом работы транслятора с языка Ассемблер являются листинг и объектный код. Причем, если объектный код формируется только при отсутствии ошибок в тексте исходной программы, то листинг создается в любом случае. Листинг содержит не только исходный текст, но и, при отсутствии ошибок, машинный код команды в шестнадцатиричном формате, а также шестнадцатиричный адрес команды. Если же в исходном тексте команды есть ошибки, то, очевидно, объектный код команды не может быть сформирован, и листинг содержит подробную информацию об ошибках.

Общая схема обработки данных транслятором приведена на рисунке 4. Стрелками на рисунке представлены информационные и логические связи между блоками транслятора.

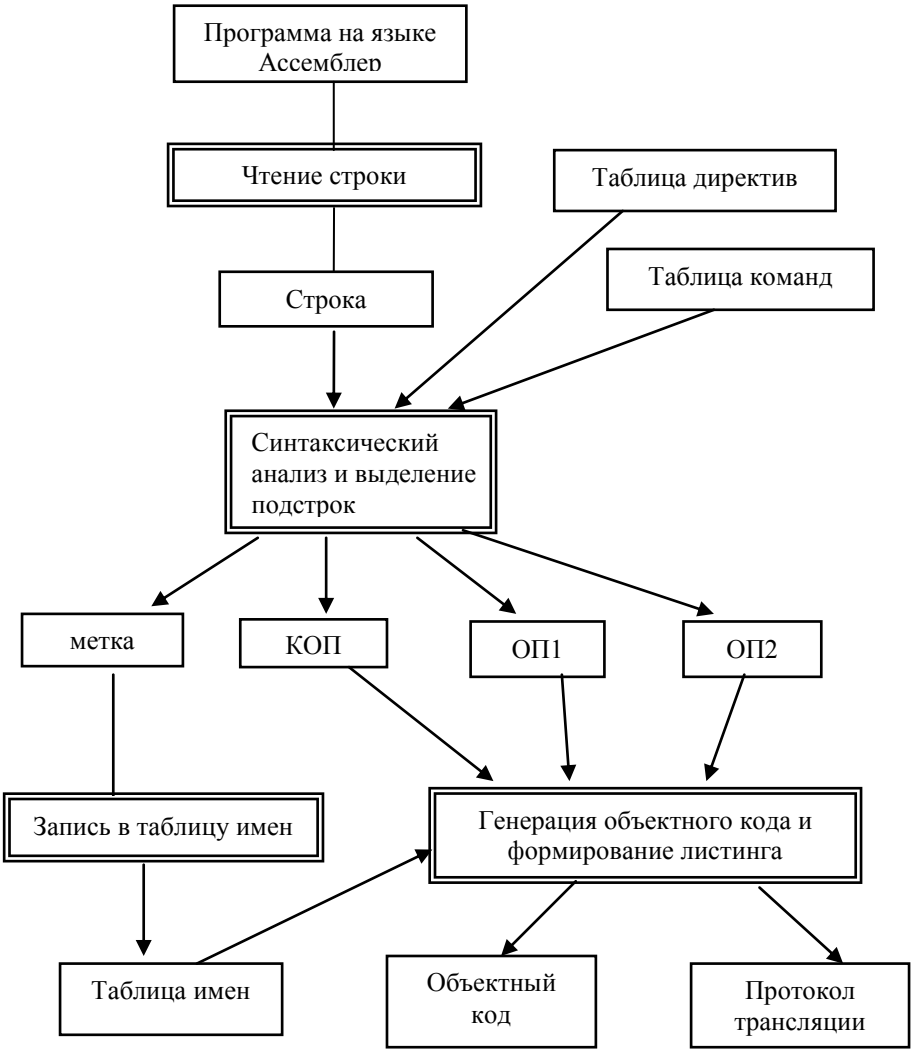


Рисунок 4 – Схема обработки данных транслятором

КОП – мнемонический код операции;
ОП1 и ОП2 – операнды первый и второй соответственно.

1.2. ТРЕБОВАНИЯ К ОТЛАДЧИКУ

Современные системы программирования, как правило, имеют в своем составе специальные средства для облегчения процесса отладки программ, называемые отладчиками.

Отладчики позволяют управлять процессом исполнения отлаживаемой программы и получать информацию о ее текущем состоянии или о содержимом памяти, изменять его, приостанавливать и вновь возобновлять исполнение программы.

Все отладчики можно разделить на две группы: универсальные отладчики для работы на уровне машинного языка и специализированные отладчики, ориентированные на работу с определенным языком программирования высокого уровня. Очевидное достоинство универсальных отладчиков состоит в том, что с помощью одного и того же отладчика можно анализировать любую исполняемою программу. Недостатком же таких отладчиков является то, что от программиста требуется определенное знание машинного языка и способов адресации памяти.

Основное достоинство отладчиков языков высокого уровня состоит в том, что ими гораздо легче пользоваться, т.к. доступ к переменной можно получить по ее имени, не задумываясь о том, с каким физическим адресом памяти она связана. Кроме того, использование отладчиков этого типа не требует знаний, в какие машинные коды преобразуется тот или иной оператор исходной программы. Недостатком же таких отладчиков является то, что они зависят от реализации самого языка высокого уровня. Как правило, они менее универсальны, поскольку не обеспечивают анализ машинных кодов, генерируемых компилятором.

Как уже отмечалось выше, в настоящей работе рассматривается задача разработки универсального отладчика – эмулятора.

На вход универсального отладчика поступает объектный код программы в заданном формате (см. п.2), и отладчик должен обеспечивать проверку правильности структуры объектного кода и выполнение команд программы. Кроме того, отладчик должен фиксировать ошибки исполнения команд (например, деление на ноль) и отслеживать недопустимые ссылки (передачу управления за пределы программы).

Схема обработки данных отладчиком представлена на рисунке 5. Стрелками на рисунке обозначены информационные и логические связи между отдельными блоками отладчика.

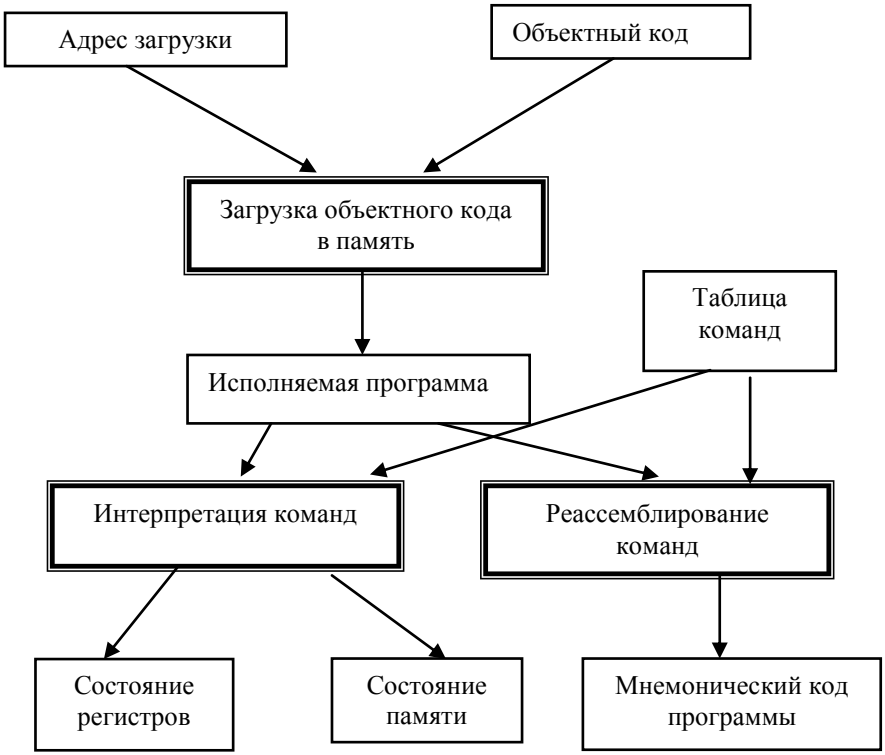


Рисунок 5 – Схема обработки данных отладчиком.

2. ЗАДАНИЕ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

Разработать программу транслятора для подмножества команд языка Ассемблера процессора *i8086/8088*. Транслируемая программа имеет следующий формат:

```
NAMESEG     SEGMENT
        ORG   выражение
        ...
        область команд
        ...
NAMEDATA    DB
        DW
        ...
        область данных
        ...
NAMESEG     ENDS
        END
```

В качестве признака завершения вычислений использовать команду INT 20H. В результате работы транслятора должны быть сформированы протокол трансляции и объектный код программы. Протокол трансляции представляется файлом-листингом, в котором содержится объектный код и мнемонический код команд, сообщения об ошибках, информация о результатах трансляции и т. д. Объектный код программы записывается в файл объектного кода и должен иметь следующий формат:

H запись-заголовок
T тело
E запись-конец

Здесь

запись-заголовок имеет формат:

H<Имя_сегмента><Длина_кода>

запись-тело:

T<адрес_кода><длина_кода><Код>

запись-конец:

E<точка_входа>

Коды команд и данных записать в виде строки, представляющей шестнадцатеричную константу.

Во всех вариантах реализовать обработку команды MOV и команд, определенных номером варианта (п. 4). Кроме того, во всех вариантах предусмотреть обработку директив:

1) **SEGMENT, ENDS, END;**
2) **ORG, OFFSET;**
3) **DB;**
4) **DW.**

Разработать среду отладчика-эмулятора, в которой предусмотреть следующие возможности:

1) загрузка объектного кода в память;
2) вывод состояния регистров, регистра флагов и заданной области памяти;
3) вывод реассемблированного текста программы;
4) пошаговое выполнение программы.

Исходными данными для отладчика являются объектный код и значения сегментных регистров. Результат работы отладчика – состояния регистров и оперативной памяти после выполнения каждой команды, отображаемые на экране компьютера.

3. ПОРЯДОК ВЫПОЛНЕНИЯ КУРСОВОЙ РАБОТЫ

1) Получить вариант задания у преподавателя. Студенты дневной формы обучения получают один вариант на бригаду из двух человек (один студент разрабатывает транслятор, другой – отладчик). Студенты заочной формы обучения разрабатывают транслятор и выбирают вариант задания по формуле:

n = xx mod 50, где ***xx*** − последние цифры зачетной книжки.

Заполнить

2) Разработать техническое задание (ТЗ) на проектирование в соответствии с вариантом задания. В ТЗ необходимо указать назначение программы, реализуемые ею функции, входные и выходные данные и их форматы, особые ситуации и реакцию разрабатываемой программы на них.

3) Определить форматы кодирования команд, заданных в варианте. Форматы кодирования команд должны соответствовать форматам команд микропроцессора *i8086 /8088*.

4) Разработать метод решения задачи. Рекомендуется в описании метода привести схему обработки данных. В описании схемы указать назначение блоков данных и содержащуюся в них информацию, а также выполняемые преобразования для блоков действий.

5) Разработать структуры данных. На основе схемы обработки данных определить способы и форматы хранения данных. Указать типы, размеры, возможные значения для переменных, записей, таблиц и т.д.

6) Разработать алгоритм решения задачи. Используя схему обработки данных и описание структур данных, составить алгоритм обработки данных. Рекомендуется представить словесное описание алгоритма. При представлении алгоритма целесообразно использовать метод пошаговой детализации: на первом уровне – общий алгоритм, на каждом следующем уровне более подробно раскрываются пункты предыдущего уровня. Для отладчика должны быть приведены алгоритмы исполнения команд и формирования флагов.

7) Разработать и отладить программу. Для отладки программы необходимо определить перечень контролируемых ситуаций. В соответствии с этим перечнем, разработать тестовые примеры. В пояснительной записке необходимо привести все тесты с описанием проверяемых ими контрольных ситуаций и возможную реакцию программы.

8) Оформить пояснительную записку.

Содержание пояснительной записки :

Введение

1. Постановка задачи.
2. Определение форматов команд.
3. Разработка метода решения.

4. Выбор и обоснование структур данных.
5. Алгоритм решения задачи.
6. Описание программы.
   6.1. Назначение программы.
   6.2. Требования к программному и техническому обеспечению.
   6.3. Логическая структура программы.
   6.4. Используемые переменные и типы.
   6.5. Спецификация процедур и функций.
   6.6. Сообщения, выдаваемые программой.
   6.7. Вызов и загрузка программы.
7. Текст программы.
8. Тестирование программы.
   8.1. Разработка тестовых примеров.
   8.2. Результаты тестирования.

Заключение.
Библиографический список.
Приложения.

Допускается добавление новых и объединение отдельных разделов и подразделов. Разделы 1−5 пояснительной записки представляют собой результаты выполнения действий, предписанных пунктами 1−7. Раздел 6 должен содержать описание программы. Здесь необходимо привести описание основных типов и переменных, спецификации на процедуры, граф связи подпрограмм.

Защита курсовой работы осуществляется в соответствии с графиком, который объявляется за 14 дней до начала защиты курсовых работ. Пояснительную записку необходимо представить для проверки преподавателю за три дня до защиты. Обязательным условием выполнения работы является стыковка разработанных программных продуктов (транслятора и отладчика), их комплексная отладка и демонстрация совместной работы на тестовых примерах.

4. ВАРИАНТЫ ЗАДАНИЙ

Разрабатываемая программа (транслятор или отладчик) должна обрабатывать следующие команды: команду MOV и по одной команде из трех групп команд. Варианты заданий приведены в таблице 3, а группы команд − в таблице 4.

Команда MOV и команда первой группы должны иметь четыре формата:

КОП Р,Р
КОП Р,НО
КОП Р,ОП
КОП ОП,Р

где **КОП** − мнемонический код команды,
Р − регистр общего назначения,
НО − непосредственный операнд,
ОП − оперативная память.

Режим адресации данных в оперативной памяти задан буквой: **П** – прямая, **К** – косвенная, **И** – индексная, Б – базовая, **С** – базово-индексная.

В команде второй группы операндом может быть регистр общего назначения (**Р**) или данные в оперативной памяти (**ОП**). Режим обращения к оперативной памяти такой же, как для команды первой группы.

Например, в варианте 45 задано следующее множество команд:

MOV Р,Р
MOV Р,НО
MOV Р,ОП(базовая)
MOV ОП(базовая),Р
ADD Р,Р
ADD Р,НО
ADD Р,ОП(базовая)
ADD ОП(базовая),Р
MUL ОП(базовая)
JB ссылка
INT 21H

Таблица 3 – Варианты заданий

№ п/п	Группа 1	Группа 2	Группа 3	№ п/п	Группа 1	Группа 2	Группа 3
0	6п	4р	6	25	9и	2р	3
1	7к	7оп	8	26	2п	1оп	11
2	5с	8оп	9	27	3к	6р	9
3	9п	7р	11	28	4п	4оп	14
4	10к	4оп	12	29	5и	2оп	3
5	8и	6р	9	30	7б	3оп	4
6	1п	2оп	5	31	10б	6р	10
7	2к	4р	7	32	9с	10оп	5
8	3и	1оп	13	33	2б	7оп	6
9	5п	6оп	14	34	5б	3р	13
10	6б	5оп	13	35	3с	8р	9
11	9б	5р	3	36	7п	5р	7
12	7с	10р	4	37	8к	5оп	5
13	10с	8оп	2	38	10п	2р	2
14	3б	3оп	2	39	7и	1р	3
15	1с	9р	13	40	10и	7р	8
16	4с	9оп	14	41	1к	6р	13
17	1и	3оп	10	42	2и	3оп	12
18	3п	5оп	7	43	4к	3р	3
19	4и	7оп	7	44	8б	7оп	6
20	5к	5р	8	45	1б	4оп	11
21	6к	6оп	10	46	4б	4р	9
22	8п	1р	4	47	6с	8р	14
23	9к	7оп	6	48	2с	10р	11
24	6и	2р	1	49	8с	9р	8

Таблица 4 – Группы команд

№ п/п	Группа 1	Группа 2	Группа 3
1	ADD	INC	JP
2	ADC	DEC	JNZ
3	SUB	NOT	LOOP
4	SBB	MUL	JZ
5	CMP	IMUL	JG
6	XOR	DIV	JGE
7	AND	IDIV	JE
8	OR	SHL	JLE
9	XCHG	SHR	JA
10	TEST	SAR	JAE
11			JB
12			JBE
13			JS
14			JC

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Абель П. Ассемблер. Язык и программирование для IBM PC / П. Абель. – Минск: КОРОНА-Век, 2009. – 736с.
2. Дао Л. Программирование микропроцессора 8088 / Л. Дао – М.: Мир, 1988.– 355с.
3. Бек Л. Введение в системное программирование / Л. Бек – М.: Мир, 1988. – 448с.
4. Пильщиков В.Н. Программирование на языке Ассемблера IBM PC/ В.Н.Пильщиков. – М.: Диалог – МИФИ, 2014. – 288 с.
5. Донован Дж. Системное программирование / Дж. Донован. –М.: Мир,1975.– 540с.
6. Баррон Д. Ассемблеры и загрузчики / Д. Баррон – М.: Мир, 1974. – 74с.
7. Лебедев В. Н. Введение в системы программирования / В.Н.Лебедев. ─ М.: Статистика, 1975. – 311с.

Заказ №_____от «___»_____20__ ____Тираж_____экз.
Изд-во СевГУ