Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебная дисциплина «Встраиваемые микропроцессорные системы»

# Инструкция

по выполнению лабораторной работы
«Программирование на языке Ассемблер. Изучение средств разработки»

Минск

2017

**Лабораторная работа № 3**

**Тема работы: «Программирование на языке Ассемблер. Изучение средств разработки»**

1. **Цель работы**:

Изучение среды программирования на языке ассемблера, инструментальных средств трансляции, компоновки и отладки. Изучение базовой системы команд и базовых видов адресации. Разработка программ с использованием основных команд ассемблера.

## 2. Задание

Написать, отладить и выполнить простейшую программу на языке ассемблера. Доработать программу согласно заданию.

## 3. Оснащение работы

Техническое задание, ПК.

## 4. Основные теоретические сведения

Написание ассемблерных программ требует знания организации всей системы компьютера.

Основные элементы аппаратных средств компьютера: системный блок, клавиатура, устройство отображения, дисководы, печатающее устройство (принтер) и др. Системный блок состоит из системной платы, блока питания и разъемов расширения для дополнительных плат. На системной плате размещены: микропроцессор (МП); постоянная память (ПЗУ); оперативная память (ОП). Разъемы расширения (слоты) обеспечивают подключение устройств отображения, каналов телекоммуникаций, дополнительной памяти и др.

Процессор выполняет обработку всех команд и данных, используя 16-битовые регистры. Процессор можно представить в виде двух частей (рисунок 3.1): операционного устройства (ОУ) и шинного интерфейса (ШИ). ОУ выполняет команды, тогда как ШИ подготовляет команды и данные для выполнения. ОУ содержит арифметико-логическое устройство (АЛУ), устройство управления (УУ) и десять регистров. Три элемента шинного интерфейса – устройство управления шиной, очередь команд и сегментные регистры – выполняют три важные функции:

- ШИ управляет передачей данных на операционное устройство, в память и на внешние устройства ввода-вывода;
- четыре сегментных регистра управляют адресацией памяти объемом до 1 Мбайт;
- ШИ выполняет выборку команд.

Команды программы находятся в памяти, поэтому ШИ должен иметь доступ к ним для выборки в очередь команд. Очередь имеет размер четырех или более байт (в зависимости от процессора), поэтому ШИ должен «заглядывать вперед» и выбирать команды так, чтобы обеспечить непустую очередь команд, готовых для выполнения. ОУ и ШИ работают параллельно, причем ШИ опережает ОУ на один шаг. ОУ сообщает в ШИ о необходимости доступа к данным в памяти или к устройству ввода-вывода. Кроме того, ОУ запрашивает машинные команды из очереди команд. Пока ОУ занято выполнением первой в очереди команды, ШИ выбирает из памяти следующую команду. Эта выборка происходит во время выполнения текущей команды, что повышает скорость обработки.

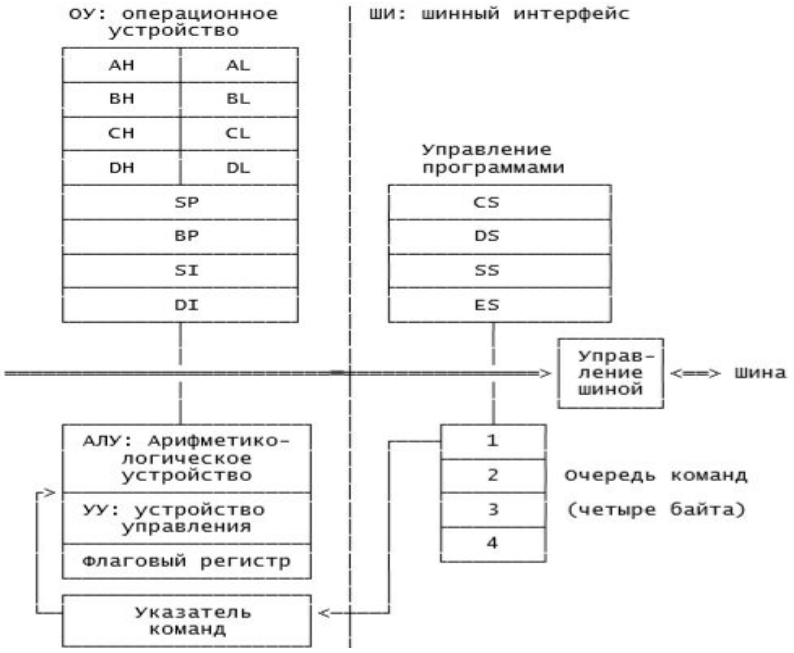



*Рисунок 3.1 – Операционное устройство и шинный интерфейс*

При выполнении программы сама программа и данные для нее хранятся в оперативной памяти (ОП). Она состоит из последовательности ячеек, к которым можно обращаться как к байтам и словам. Ячейки имеют уникальные номера – адреса. Для упрощения работы и сокращения рабочей длины адреса ОП разделяется на сегменты. Сегментом называется область, которая начинается на границе параграфа (по адресу, кратному 16-ти). Он может располагаться в любом месте ОП и иметь размер до 64 Кбайт. Смещение адреса внутри сегмента хранится в одном из регистров смещения. Типичная программа содержит три главных сегмента: сегмент кода, сегмент данных, сегмент стека.

Сегмент кодов содержит машинные команды. Обычно первая выполняемая команда находится в начале этого сегмента. Операционная система передает управление по адресу этого сегмента для выполнения программы. Сегмент кодов адресуется регистром сегмента кодов – CS. Сегмент данных содержит определенные данные, константы и рабочие области, нужные программе. Сегмент данных адресуется регистром сегмента данных – DS. Сегмент стека. Стек содержит адреса возврата: в случае программы – для возврата в ОС; в случае подпрограммы – для возврата в главную программу. Сегмент стека адресуется регистром сегмента стека – SS. Четвертый сегментный регистр – регистр дополнительного сегмента (ES) – предназначен для специального использования. Например, он используется в некоторых операциях над цепочками символов.

На рисунке 3.2 представлены регистры SS, DS и CS. Практически последовательность регистров и сегментов может быть иной. Три сегментных регистра содержат начальные адреса соответственных сегментов.

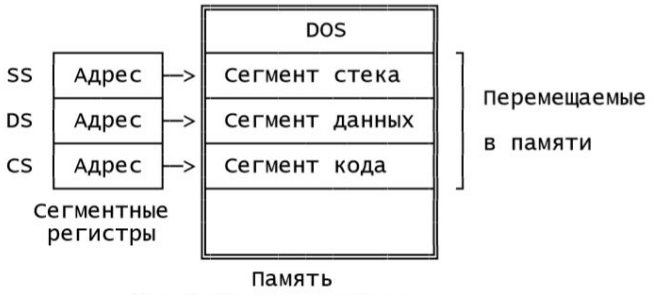



*Рисунок 3.2 – Сегменты и регистры*

Сегменты могут перекрываться и даже совпадать, потому что часто их требуемый размер меньше максимального. Пример размещения сегментов в памяти (рисунок 3.3):

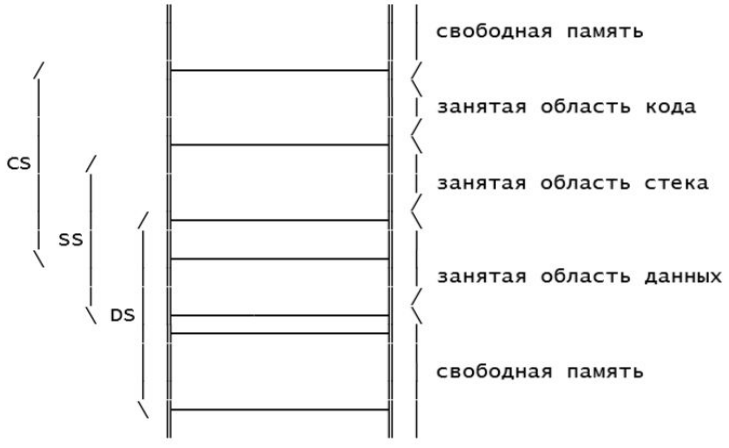



*Рисунок 3.3 – Пример размещения сегментов в памяти*

Каждый сегментный регистр обеспечивает адресацию 64К памяти, которая называется текущим сегментом. Сегменты выравнены по границам

параграфов, так что адрес в сегментном регистре предполагает наличие справа четырех нулевых битов:

- CS – регистр сегмента кода. Содержит начальный адрес сегмента кода; этот адрес плюс величина смещения в указателе команд (IP) определяет адрес команды, которая подлежит выполнению. В обычных программах нет необходимости делать ссылки на регистр CS.
- DS – регистр сегмента данных. Содержит начальный адрес сегмента данных. Этот адрес плюс величина смещения, определенная в команде, указывают на конкретную ячейку в сегменте данных.
- SS – регистр сегмента стека. Содержит начальный адрес сегмента стека.
- ES – регистр вспомогательного сегмента. Используется некоторыми операциями над цепочками для управления адресацией памяти. При этом регистр ES связан с индексным регистром DI. Если нужно использовать регистр ES, программа должна его инициализировать.

Процессор использует 14 основных регистров, применяемых для управления выполняемой программой, для адресации памяти и для обеспечения арифметических вычислений. Каждый регистр имеет длину в одно слово (16 бит) и адресуется по имени. Биты регистра нумеруются слева направо.

**Регистры общего назначения (РОН).** Основные рабочие регистры, постоянно используемые в программе. Их можно адресовать как одно целое слово либо как однобайтовую часть. Левый байт являет старшую часть слова (high), правый – младшую часть (low). Например, двухбайтовый регистр CX состоит из двух однобайтовых регистров CH и CL; ссылки на регистр возможны по любому из этих трех имен. Все регистры общего назначения можно использовать для сложения и вычитания восьмибитовых и шестнадцатибитовых значений:

- Регистр AX – основной сумматор. Применяется для всех операций ввода-вывода, некоторых операций над цепочками, для многих арифметических операций. Так, команды умножения, деления и сдвига предполагают его обязательное использование.
- Регистр BX – базовый регистр, единственный регистр общего назначения, который можно использовать как индексный для расширенной адресации. Также применяется для вычислений.
- Регистр CX – счетчик. Необходим для управления числом повторений циклов, для операций сдвига. Также используется для вычислений.
- Регистр DX – регистр данных. Применяется для некоторых операций ввода-вывода, для операций умножения и деления над такими числами, которые используют регистровую пару DX:AX.

**Регистры-указатели** (регистровые указатели) обеспечивают доступ к данным в сегменте стека. Иногда используются для операций сложения и вычитания. Основное назначение:

- регистр SP – указатель стека: обеспечивает использование стека в памяти, позволяет временно хранить адреса и данные; при этом связан с регистром SS для адресации стека.
- Регистр BP – указатель базы; облегчает доступ к параметрам – данным и адресам, переданным через стек.

**Индексные регистры** используются для расширенной адресации и в операциях сложения и вычитания. Основное назначение:

- регистр SI – индекс источника: применяется для некоторых операций над цепочками, при этом связан с регистром DS.
- регистр DI – индекс назначения: тоже применяется для цепочечных операций, при этом связан с регистром ES.

**Регистр указателя команд** (регистр командного указателя, счетчик команд). Регистр IP. Содержит смещение (относительный адрес) для команды, подлежащей выполнению. Обычно в программе не используется. Его значение можно изменять при использовании отладчиков для отладки программы.

**Регистр признаков** (регистр флагов, флаговый регистр). Девять из его 16-ти битов активны, определяют текущее состояние процессора и результатов выполнения. Многие арифметические команды и команды сравнения изменяют состояние некоторых признаков (флагов).

При загрузке программы сегменты размещаются в памяти, как показано на рисунке 3.4.

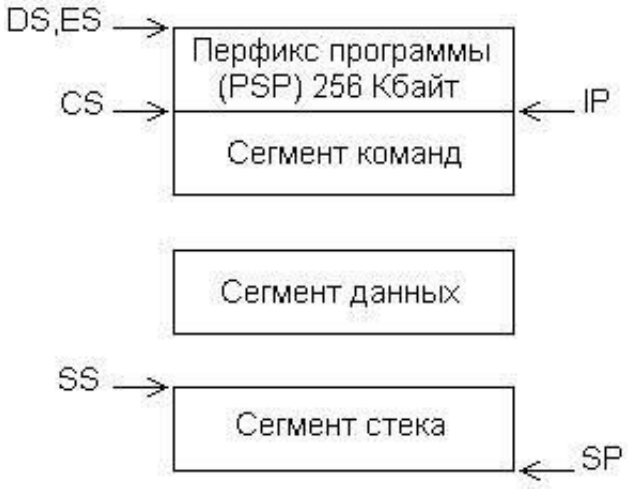



*Рисунок 3.4 – Образ программы в памяти*

Образ программы в памяти начинается с префикса программного сегмента (ППС, Program Segment Prefics – PSP), образуемого и заполняемого системой. ППС всегда имеет размер 256 байт; он содержит таблицы и поля данных, используемые системой в процессе выполнения программы. Вслед за ППС располагаются сегменты программы в том порядке, как они объявлены в программе. Сегментные регистры автоматически инициализируются следующим образом: ES и DS указывают на начало ППС (что позволяет, сохранив их содержимое, обращаться затем в программе к ППС), CS – на

начало сегмента команд, а SS – на начало сегмента стека. В указатель команд IP загружается относительный адрес точки входа в программу (из операнда директивы END), а в указатель стека SP – величина, равная объявленному размеру стека, в результате чего указатель стека указывает на конец стека (точнее, на первое слово за его пределами). Так после загрузки программы в память адресуемыми оказываются все сегменты, кроме сегмента данных. Инициализация регистра DS в первых строках программы позволяет сделать этот сегмент адресуемым. Рисунок 3.4 подчеркивает важнейшую особенность архитектуры процессоров Intel: адрес любой ячейки памяти состоит из двух слов, одно из которых определяет расположение в памяти сегмента, другое – смещение в пределах сегмента. Сегмент всегда начинается с адреса, кратного 16, т.е. на границе 16-байтового блока памяти – параграфа. Сегментный адрес можно рассматривать как номер параграфа, с которого начинается сегмент. Размер сегмента определяется объемом содержимых в нем данных, но не может превышать 64 Кбайт, что определяется максимально возможной величиной смещения.

Сегментный адрес сегмента команд хранится в регистре CS, а смещение к адресуемому байту – в указателе команд IP. После загрузки программы в IP заносится смещение первой команды программы; процессор, считав ее из памяти, увеличивает содержимое IP точно на длину этой команды (команды имеют длину от 1 до 6 байт), в результате чего IP указывает на вторую команду программы. Выполнив первую команду, процессор считывает из памяти вторую, опять увеличивая значение IP. В результате в IP всегда находится смещение очередной команды, следующей за выполняемой. Это правило нарушается лишь при выполнении команд переходов, вызовов подпрограмм и обслуживания прерываний.

Сегментный адрес сегмента данных обычно хранится в регистре DS, а смещение может находиться в одном из регистров общего назначения, например, в BX или SI.

Программа есть упорядоченная последовательность команд, необходимых компьютеру для решения поставленной задачи. В ходе создания программы можно выделить следующие этапы:

- постановка задачи;
- математическое описание (для научно-инженерных задач);
- алгоритмизация вычислительного процесса;
- составление программы;
- отладка программы;
- решение задачи на компьютере и анализ результатов.

Постановка задачи. Формулируется цель решения задачи и подробно описывается ее содержание. Анализируются характер и сущность всех величин, используемых в задаче, и определяются условия, при которых она решается.

Математическое описание. Характеризуется математической

формализацией задачи, при которой существующие соотношения между величинами, определяющими результат, выражаются посредством математических формул. Так формируется математическая модель явления с определенной точностью, допущениями и ограничениями.

Алгоритмизация вычислительного процесса. Составляется алгоритм решения задачи. Процесс обработки данных разбивается на отдельные относительно самостоятельные блоки, и устанавливается последовательность выполнения блоков. Разрабатывается блок-схема алгоритма.

Составление программы. Алгоритм решения задачи переводится на конкретный язык программирования.

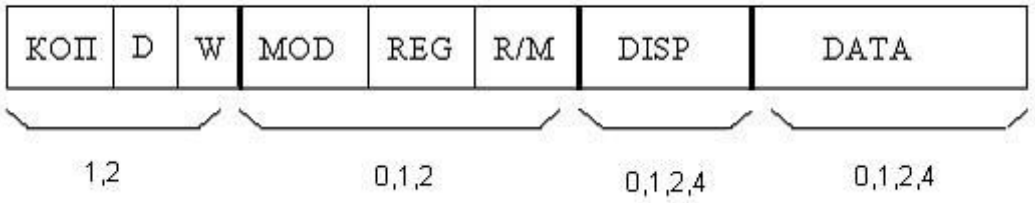
Отладка программы заключается в поиске и устранении синтактических и логических ошибок в программе. В ходе синтактического контроля программы транслятором выявляются конструкции и сочетания символов, недопустимые с точки зрения правил их построения или написания, принятых в данном языке. Сообщения об ошибках выдаются программисту, причем вид и форма выдачи сообщений зависят от вида языка и версии используемого транслятора. После устранения синтактических ошибок проверяется логика работы программы в ходе ее выполнения с конкретными исходными данными. Для этого используются специальные методы, например, в программе выбираются контрольные точки, для которых вручную рассчитываются промежуточные результаты. Эти результаты сверяются со значениями, получаемыми в этих точках при выполнении отлаживаемой программы. Кроме того, для поиска ошибок можно использовать отладчики, выполняющие при отладке специальные действия, такие как удаление, замена или вставка отдельных операторов или целых фрагментов программы, вывод или изменение значений заданных переменных.

Решение задачи на компьютере и анализ результатов. После отладки программы ее можно использовать для решения прикладной задачи.

Язык ассемблера – это язык, предназначенный для представления в удобочитаемой символической форме программ, записанных на машинном языке. Он позволяет пользоваться определенными кодами операций, присваивать удобные имена ячейкам и областям памяти, также задавать удобные схемы адресации.

Команда машинной программы (машинная команда) есть элементарная инструкция машине, выполняемая автоматически без дополнительных указаний. Она состоит из двух частей – операционной и адресной. Операционная часть есть группа разрядов в команде, предназначенная для представления кода операции. Адресная часть команды есть группа разрядов, где записываются коды адреса (адресов), по которым хранятся операнды. Адресная часть не всегда имеется. Обобщенные форматы машинных команд различаются адресной частью (количеством адресов).

Обобщенная структура машинной команды:




Длина команды равна 1-6 байт.

Поле D – направление повторяющейся операции: D = 0 – слева направо; D = 1 – справа налево.

Поле MOD – режим адресации.

Поле R/M – регистр или память. Эти поля определяют один из 24 способов формирования относительного адреса операндов внутри сегмента.

Поле W формат – данных: W = 0 – байт; W = 1 – слово.

Поле REC – адрес регистра для операнда.

Поле DISP – смещение.

Поле DATA – непосредственные данные.

По видам выполняемых операций команды делятся на следующие группы:

- операции пересылки информации;
- арифметические операции;
- логические операции;
- операции над цепочками символов (строковые операции);
- операции обращения к ПУ;
- операции передачи управления;
- операции обслуживающие и вспомогательные.

Операции пересылки информации:

MOV – переслать;

XCHG – обменять (переставить);

IN – ввести в порт;

OUT – вывести из порта;

PUSH – занести в стек;

POP – извлечь из стека.

Арифметические операции. Их операнды бывают двоичные (целые 8, 16 или 32бит, со знаком или без знака), двоично-десятичные (от 1 до 255 байт, в упакованном или распакованном формате). Сюда входят:

- команды сложения, вычисления и сравнения (ADD, SUB, CMP);
- команды приращения (INC – инкремент, DEC – декремент);
- команды умножения (MOL, IMUL);
- команды деления (DIV, IDIV);
- команды преобразования форматов.

Логические операции служат для сравнения, сбора и установки битов операнда для преобразования кодов, и др.:

- логические операции (OR, AND, NOT, XOR, TEST);

- линейные (арифметические и логические) и циклические сдвиги влево и вправо на один или несколько разрядов.

Операции над цепочками символов обеспечивают обработку (пересылку, сравнение, загрузку, сканирование, повторение) одного элемента цепочки за одно обращение к памяти. За счет автоматического повторения они позволяют обрабатывать цепочки длиной до 64 Кбайт значительно быстрее, чем с помощью программных циклов.

Операции обращения к периферийным устройствам выполняются с помощью обработки прерываний. Основные команды:

- INT – различные программные прерывания;
- INTO – условное прерывание при переполнении;
- IRET – возврат из прерывания.

Операции передачи управления (перехода). Операции безусловной передачи управления – всегда задают выполнение не очередной команды программы, но той команды, адрес которой указан в команде перехода. Операции условной передачи управления – действуют аналогично, но лишь в том случае, когда выполняется заданное условие перехода. При невыполнении условия команда ничего не делает. В числе этих операций:

- команда безусловного перехода JMP;
- набор команд условие переходов J*;
- команда вызова процедуры CFLL
- команда возврата из процедуры RET;
- команды управления циклом LOOP.

Операции обслуживания и вспомогательные служат для управления МП. Сюда входят:

- команды установки и сбора признаков;
- команда состава HLT;
- команда ожидания WAIT;
- команда выхода ESC;
- команда холостого хода NOP.

На уровне языка ассемблера программист взамен машинных кодов пользуется символическими мнемокодами и описательными именами для полей данных и адресов памяти. Программа, написанная такими мнемокодами, есть исходный модуль. Для его написания применяется любой подходящий экранный редактор. Затем с помощью программы ассемблера исходный текст компилируется (точнее, ассемблируется) в машинный код, известный как объектная программа. Потом программа упаковщика LINK определяет все адресные ссылки для объектной программы и генерирует загрузочный модуль.

**Комментарии.** Использование комментариев улучшает ясность программы, особенно если назначение набора команд непонятно. Комментарий всегда начинается символом точка с запятой; ассемблер полагает, что все символы строки после него составляют комментарий. Комментарий может

содержать любые печатные символы, включая пробел; он может занимать всю строку или следовать за командой на той же строке. Комментарии появляются только в исходных текстах и в распечатках ассемблирования исходного модуля; они не приводят к генерации машинных кодов, поэтому можно использовать любое количество комментариев без влияния на эффективность выполнения программы. В дальнейшем команды языка ассемблера представлены заглавными буквами, комментарии – строчными (только для удобочитаемости).

**Основной формат кодирования команд** имеет вид:

[метка] команда [операнд(ы)]

**Метка** (если имеется), команда и операнд (если имеется) разделяются хотя бы одним пробелом либо символом табуляции. Максимальная длина строки – 132 символа, но предпочтительно работать со строками в 80 символов (соответственно ширине экрана). Метка, команда и операнд не обязательно должны начинаться с определенной позиции строки. Но рекомендуется записывать их в колонку – для лучшей удобочитаемости программы.

**Метки м**огут содержать следующие символы:

буквы: от A до Z и от a до z цифры: от 0 до 9;

спецсимволы: знак вопроса “?”; точка “.” (только первый символ);

знак “@”;

подчеркивание “_”;

знак доллара “$”.

Первым символом метки должна быть буква или спецсимвол. Заглавные и строчные буквы не различаются. Максимальная длина метки – 31 символ. Обычно метки заканчиваются двоеточием. Следует использовать описательные и смысловые метки. Имена регистров зарезервированы, их можно использовать лишь для указания регистров.

**Команды с**пецифицируют выполняемое действие. В сегменте данных команда-директива определяет поле, рабочую область или константу. В сегменте кода команда определяет действие, например, пересылка (MOV) или сложение (ADD).

**Операнды.** Команда может иметь один либо два операнда, или вообще не иметь операндов. Операнды определяют:

- начальное значение данных;
- элементы, над которыми выполняет действие команда.

**Директивы (псевдокоманды)** являют ряд операторов, которые позволяют управлять процессом ассемблирования и формирования распечатки. Они действуют только в ходе ассемблирования программы и не генерируют машинных кодов. Кроме прочих существуют директивы управления распечаткой, управляющие форматом распечатки (листинга).

Обычно язык ассемблера поставляется без среды разработки программ.

Поэтому разработчику приходится самостоятельно контролировать весь процесс создания программы, что предполагает такие этапы:

- Постановка задачи – точное и подробное описание функциональности будущей программы, а также описание всех входных и выходных данных и способа их передачи программе.
- Разработка алгоритма программы – построение блок-схемы или граф-схемы, либо текстовое или математическое описание решения.
- Формализация алгоритма – запись алгоритма на языке программирования. Создание текстового файла программы с расширением ASM. Отсутствие среды разработки позволяет программисту самостоятельно выбрать текстовый редактор для написания кода программы. Для этой цели подойдет любой текстовый редактор с нумерацией строк.
- Ассемблирование программы – перевод программы из текстового вида в машинный код. При использовании ассемблера фирмы Borland необходимо выполнить команду вроде TASM MY.ASM (предполагается условное имя исходного файла MY.ASM), то есть запустить ассемблер Tasm и передать с командной строки имя файла, содержащего программу. Если программа имеет синтактические ошибки, ассемблер выдаст сообщение об ошибке с указанием номера строки и описанием каждой ошибки (нужно вернуться на этап 3 и исправить ошибки). При успешном ассемблировании будет создан файл, содержащий объектный код программы MY.OBJ, который еще не будет исполняемым модулем.
- Компоновка программы. Компоновка – создание из файла объектного кода исполняемого модуля командой TLINK MY.OBJ. Как параметр компоновщик TLINK принимает имя объектного файла (здесь – MY.OBJ). При успешной компоновке создается исполняемый модуль MY.EXE.
- Запуск и тестирование исполняемого модуля программы. На этом этапе необходимо проверить, соответствует ли написанная программа постановке задачи, сделанной на этапе 1. Неправильная работа программы говорит об алгоритмической (семантической) ошибке, поэтому для ее устранения нужно вернуться на этап разработки алгоритма (этап 2).

## 5. Порядок выполнения работы

Задание 1. Написание программы на языке ассемблера. Используя текстовый редактор, создать и отредактировать исходный модуль программы, текст которого приведен ниже.

Постановка задачи. Написать программу, выводящую на экран строку "Привет!".

Разработка алгоритма программы. Алгоритм простейший линейный, разработки не требует.

Формализация (запись) алгоритма. В текстовом редакторе создать файл PRIVET.ASM и записать в него следующий код (без номеров строк и без

комментариев):

```
 1 DATA SEGMENT                ;описание сегмента данных
 2      MES DB 'Привет!$'      ;строка вывода на экран. '$' - признак конца
строки
 3 DATA ENDS                   ;конец сегмента данных
 4
 5 CODE SEGMENT                ;начало сегмента кода
 6 START:                      ;метка начала программы
 7 ASSUME CS:CODE, DS: DATA    ;директива привязки сегментов
 8 MOV AX, DATA                ;настройка сегмента данных
 9 MOV DS, AX
10
11 MOV AH,9                    ;для вызова функции 9 - вывод строки на
экран
12 LEA DX, MES                 ;адрес выводимой строки
13 INT 21H                     ;вызов прерывания для вывода строки
14
15 MOV AX, 4C00H               ;для вызова функции завершения
программы
16 INT 21H                     ;вызов функции завершения программы
17 CODE ENDS                   ;конец сегмента кода
18 END START                   ;конец программы с точкой входа START
```

Описание программы PRIVET.ASM.

Строки 1-3 содержат описание сегмента данных – области памяти, где хранятся данные для программ.

Строки 5-7 – код программы, ее исполняемая часть.

В строках 8 и 9 выполняется настройка сегмента данных программы.

Строки 11-13 – вывод строки на экран при помощи функции 9 прерывания 21H.

Строки 15 и 16 – стандартное завершение программы.

После символа ';' пишутся комментарии, они не обрабатываются ассемблером. Ассемблирование программы – выполняется командой TASM PRIVET.ASM. Компоновка программы – выполняется командой TLINK PRIVET.OBJ. При компоновке программы появится сообщение об отсутствии сегмента стека. Но в случае столь простой программы без стека можно обойтись. Запуск и проверка – выполняется командой PRIVET.EXE.

Переход на новую строку. Для перехода на новую строку достаточно вывести на экран символы перевода строки и возврата каретки (CR/LF). Эти символы имеют коды 10 и 13. Если в программе нужно после вывода строки перейти на новую строку, то для этого достаточно переписать вторую строку

программы:

MES2 DB 'Выводим строку и переходим на новую...', 10, 13, '$'

Переход на новую строку можно выполнить до вывода сообщения на экран:

MES3 DB 10, 13, 'Выводим с новой строки...$'

Задание 2. Написать программу, которая выводит одно под другим сообщения: Привет! Я – компьютер! Для этого создайте новый исходный модуль, выполните ассемблирование и компоновку, после чего убедитесь в работоспособности программы.

## 6. Форма отчета о работе

*Лабораторная работа № ___*

*Номер учебной группы _____*

*Фамилия, инициалы учащегося _____*

*Дата выполнения работы _____*

*Тема работы: _____*

*Цель работы: _____*

*Оснащение работы: _____*

*Результат выполнения работы: _____*

_____

_____

## 7. Контрольные вопросы и задания

1. Команды и директивы Ассемблера. Формат и отличия.
2. Какова цель сегментации памяти?
3. Что такое базовый адрес сегмента?
4. Какие значения может принимать базовый адрес сегмента?
5. Каков максимальный размер сегмента и почему?
6. Из каких логических сегментов состоит исходный модуль ассемблерной программы?
7. Какими директивами описывается сегмент?
8. Как описываются различные типы данных, используемые программой?
9. В чем заключается инициализация сегментных регистров?
10. Что такое ассемблирование и компоновка программы?
11. Что представляет собой исходный модуль программы?
12. Опишите стандартное начало и окончание сегмента кодов?
13. Каково содержание файлов с расширениями *.ASM, *.LST, *.OBJ, *.MAP, *.EXE?
14. Каково назначение и в чём отличия метки команды от имени директивы?
15. Для чего требуется помечать начальную команду программы меткой?
16. Как завершается исходный модуль программы?
17. Для чего предназначена программа DOSBox?
18. Как сделать Вашу рабочую папку текущей при использовании программы

DOSBox?

19. Каким образом можно сохранить копию экрана в программе DOSBox?

## 8. Рекомендуемая литература

Финогенов, К. Г. Основы языка Ассемблера [Текст] / К. Г. Финогенов. – М.: Радио и связь, 2000.

Финогенов, К. Г. Использование языка Ассемблера [Текст]: учеб. пособие для вузов / К.Г. Финогенов. – М.: Горячая линия Телеком, 2004.

Юров, В. И. Assembler [Текст]: учеб. пособие для вузов / В. И. Юров. 2-е изд. – СПб.: Питер, 2007.