МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

КАФЕДРА «АВТОМАТИЗАЦИЯ ПРОИЗВОДСТВЕННЫХ ПРОЦЕССОВ»

ИЗУЧЕНИЕ МАШИННО-ОРИЕНТИРОВАННОГО ЯЗЫКА АССЕМБЛЕР ДЛЯ DOS И WINDOWS

Методические указания

Волгоград

2013

УДК 681.3.06(075)

Рецензент
С. И. Николаева

Печатается по разрешению редакционно-издательского совета
Волгоградского государственного технического университета

Изучение машинно-ориентированного языка Ассемблер для DOS и Windows: метод. указания / сост. В. С. Поляков ; ВолгГТУ. – Волгоград, 2013. – 17 с.

Содержат основные сведения о работе языка низкого уровня Ассемблер. Наиболее подробно рассмотрены принципы и механизмы его работы, а так же его достоинства и недостатки.

Предназначены для студентов высших технических учебных заведений, изучающих курсы «Вычислительные машины, системы и сети».

Учебное издание

Составитель Владимир Сергеевич **Поляков**

ИЗУЧЕНИЕ МАШИННО-ОРИЕНТИРОВАННОГО ЯЗЫКА АССЕМБЛЕР ДЛЯ DOS И WINDOWS

Методические указания

Темплан 2013 г. (учебно-методическая литература). Поз. № 37.
Подписано в печать 18.11.2013. Формат 60x84 1/16.
Бумага офсетная. Гарнитура Times. Печать офсетная.
Усл. печ. л. 0,93. Тираж 10 экз. Заказ

Волгоградский государственный технический университет.
400005, г. Волгоград, просп. им. В. И. Ленина, 28, корп. 1

Отпечатано в типографии ИУНЛ ВолгГТУ
400005, г. Волгоград, просп. им. В.И. Ленина, 28, корп. 7

1. ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Целью работы является изучение основ работы языка низкого уровня Ассемблер с использованием программы-эмулятора.

2. ОСНОВНЫЕ СВЕДЕНИЯ

Ассемблер – язык низкого уровня. Является по своей сути машинно-ориентированным языком, где каждая бинарная команда процессора для простоты программирования заменена именем (мнемоникой).

Как и у других языков, у ассемблера есть как преимущества, так и недостатки. Достоинства заключаются в следующем:

1. Максимальное использование возможностей ЭВМ.

2. Минимальные инструментальные средства.

3. Минимальная память и минимальное время выполнения (самое высокое быстродействие).

Недостатки:

1. Трудности при программировании (необходимо на высоком уровне знать «железо»).

2. Сложности при переносе программы в систему с другим типом процессора.

Основные типы программ, которые целесообразно писать на ассемблере:

1. Программы для BIOS (ПЗУ).

2. Программы режима реального времени.

3. Драйвера периферийных устройств.

4. Программы каналов ввода-вывода.

5. Программы обработки прерываний и др.

Для того чтобы начать изучение языка Ассемблер, следует разобраться с тем, как именно он работает. Иными словами, прежде чем начать работу с полноценным языком программирования низкого уровня, надо изучить механизмы его работы. Для этого используем программу-эмулятор. Данный эмулятор ассемблера хорош тем, что позволяет с максимальной наглядностью отследить пересылку данных, работу с портами ввода/вывода, установку и сброс флагов, а также принцип функционирования стека. Все это, а также другие элементы будут разобраны в данном пособии.

Итак, начнем с того, что представим вид эмулятора при его запуске (рис. 1).

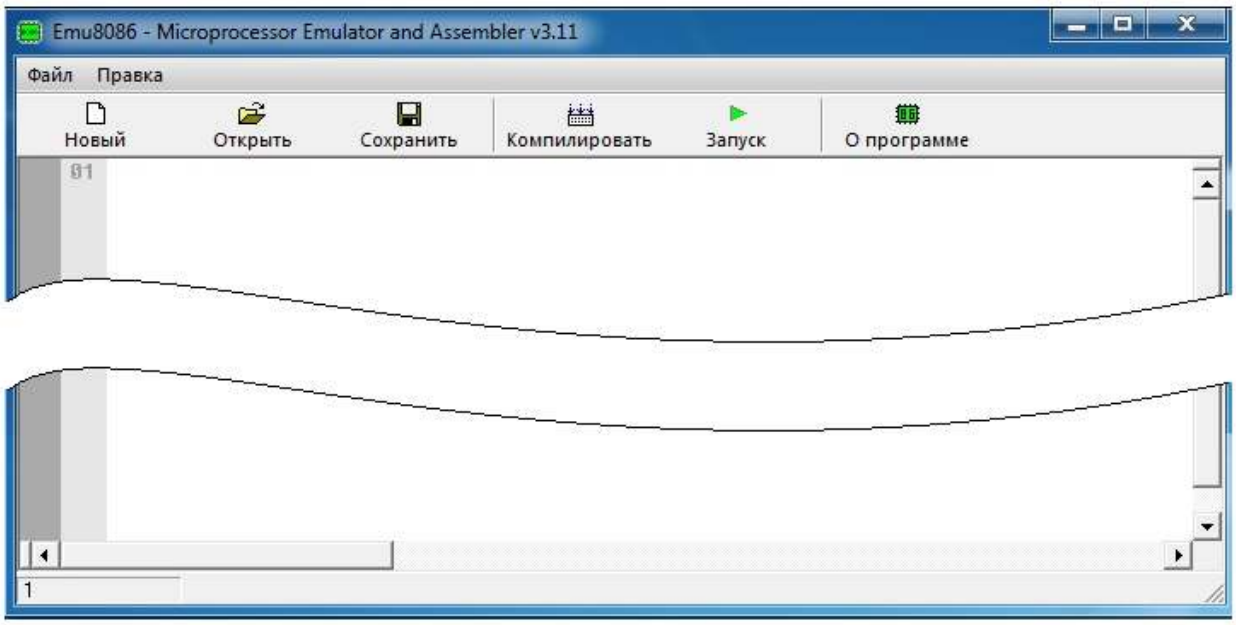

Рис. 1. Вид эмулятора при запуске

Естественно, первым делом должны будем создать новый файл, для чего нажмем соответствующую кнопку. После этого предоставляется возможность выбора (рис. 2).

При выборе файла с расширением *.com или *.exe эмулятор сам создаст шаблон, необходимый для работы в первом или втором случае. Подобное может быть удобно, если нет смысла тратить время на уже хорошо

известную процедуру. Однако сейчас делается лишь первый шаг в изучении языка, поэтому выбирает вариант «пустой», как и отмечено на рисунке 2.

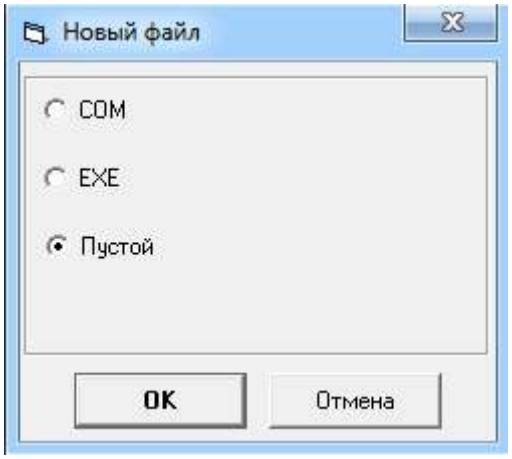

Рис. 2. Выбор типа создаваемого файла

Разберемся с форматом ассемблерной строки. Каждая строка будет иметь следующий вид:

метка команда/директива операнды ; комментарий

Следует отметить, что ни одна из составных частей строки не является обязательной. К примеру, может отсутствовать метка, присутствовать два, один или ни одного операнда. Комментарии же и вовсе используются тогда, когда это удобно разработчику программ. А вот операндов без команды или директивы не существует. В дальнейшем мы столкнемся со всеми возможными комбинациями.

Самый распространенный вариант выглядит следующим образом:

команда операнд 1, операнд 2 , комментарий

Меткой может являться набор любых букв и цифр английского алфавита, но цифра не может быть первым символом. После метки ставится двоеточие, если она стоит перед командой. Если метка стоит перед директивой, то двоеточие не ставится.

В ассемблере существует набор команд, который можно разделить на определенного рода группы. Однако почти все команды в качестве операндов используют регистры. Регистры – наиболее быстрая память из

существующих, потому как являются неотъемлемой частью процессора, то есть центра любой ЭВМ. Именно поэтому необходимо чуть подробнее разобраться с теми регистрами, которые мы будем использовать.

Регистры общего назначения: AX, BX, CX и DX. Все эти регистры могут использоваться для арифметических вычислений, операций над строками. AX и DX также используются для операций ввода/вывода. Однако у каждого из этих четырех регистров есть и уникальные особенности.

AX (Accumulator register) – это регистр-аккумулятор, временное хранилище данных. Помимо того, определенные команды более эффективны, если ссылаются на этот регистр.

BX (Base register) – базовый регистр. Его уникальное назначение состоит в том, что он единственный из регистров общего назначения используется в качестве инструмента при операциях расширенной адресации (к примеру, косвенной).

CX (Count register) – регистр-счетчик. Он необходим для работы с циклами, а также неплох при выполнении операций сдвига в качестве операнда-счетчика.

DX (Data register) – регистр данных, а также портов ввода/вывода.

Эти четыре регистра имеют отличную от прочих структуру, которую и рассмотрим на примере регистра AX. Регистры общего назначения разделяются на старшую (High) и младшую (Low) части, каждая размером 1 байт (для AX – AH и AL). Если же необходимо работать с 32-хразрядными командами, то AX расширяем до EAX (Extended AX). Общая структура EAX показана на рисунке 3. То же самое справедливо и для остальных регистров общего назначения.

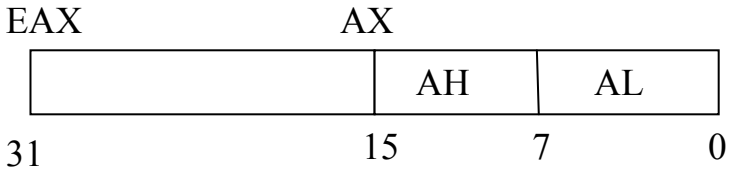

Рис. 3. Структура регистров общего назначения на примере регистра-аккумулятора AX

Помимо регистров общего назначения существуют:

– регистровые указатели: указатель стека SP (Stack Pointer) и указатель базы BP (Base Pointer); используются для указания смещения при адресации памяти;

– индексные регистры: индекс источника SI (Source Index) и индекс приемника DI (Destination Index); применяются для индексной адресации;

– сегментные регистры: регистр сегмента кода CS (Code Segment), регистр сегмента данных DS (Data Segment), регистр сегмента стека SS (Stack Segment) и регистр дополнительного сегмента данных ES (Extra Segment); используются для указания сегмента при адресации памяти;

– регистр командного указателя IP (Instruction Pointer); указывает на смещение команд в сегменте кода.

Их структура выглядит иначе, что и продемонстрируем на примере индексного регистра SI (рис. 4).

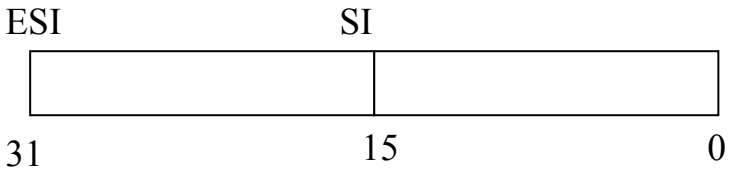

Рис. 4. Структура остальных регистров

Наконец, рассмотрим регистр флагов FLAGS. Он содержит 16 флагов размером 1 бит. Каждый флаг управляется различными командами для индикации состояния операции. Значение любого флага остается без изменений, пока не будет изменено какой-либо командой. На рисунке 5 показана структура регистра флагов (звездочками здесь отмечены неиспользуемые биты).

Номер бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Флаг	*	*	*	*	OF	DF	IF	TF	SF	ZF	*	AF	*	PF	*	CF

Рис. 5. Структура регистра флагов

Разберемся с тем, для чего предназначен каждый из используемых флагов:

– CF (Carry Flag) – флаг переноса. Содержит значение "переносов" (0 или 1) из старшего разряда при арифметических и сдвиговых операциях;

– PF (Parity Flag) – флаг четности. Нечетное число бит в результате проведенной операции (из 8 проверяемых бит) устанавливает флаг в 0, а четное в 1.

– AF (Auxiliary Carry Flag) – дополнительный флаг переноса. Устанавливается в 1, если арифметическая операция переносит 4-й справа бит в команде. Данный флаг связан с операциями над символами ASCII (таблица символов), с которой мы столкнемся в последующих лабораторных работах.

– ZF (Zero Flag) – флаг нуля, срабатывающий при работе с арифметическими операциями и командами сравнения. Ненулевой результат устанавливает флаг в 0, нулевой в 1. Налицо различие человеческой и «машинной» логики, потому что последняя тут воспринимает 0 как «нет» (результат не равен нулю), а 1 соответственно наоборот.

– SF (SIgn Flag) – флаг знака. Устанавливается после арифметических операций или операций сравнения, ориентируясь на старший бит, являющийся знаковым. Положительный результат устанавливает флаг в 0, а отрицательный в 1.

– IF (Interrupt Flag) – флаг прерывания. При нулевом состоянии этого флага прерывания запрещены, при единичном – разрешены.

– DF (DIrection Flag) – флаг направления. Используется в строковых операциях для определения направления передачи данных. При нулевом состоянии выполняемая команда увеличивает содержимое индексных регистров SI и DI, вызывая передачу данных слева направо. При нулевом – уменьшает содержимое этих регистров, вызывая передачу данных справа налево.

– OF (Overflow Flag) – флаг арифметического переполнения. Переносит вниз старшего (знакового) бита при арифметических операциях.

– TF (Trap Flag) – флаг пошагового выполнения. Если флаг установлен в 1, то процессор переходит в режим пошагового выполнения команд (в каждый момент выполняется одна команда под пользовательским управлением).

Рассмотрим работу простейшей команды пересылки данных mov, где данные из регистра bx должны быть скопированы в регистр ax.

```
mov ax, bx
```

Здесь второй операнд (регистр bx) является источником, а первый (регистр ax) – получателем. Если в bx находилось число 5, то после выполнения данной команды число 5 окажется и в ax.

Начиная работать с командами, мы обязаны разбираться в методах адресации данных. Первым делом нам необходимо уметь применять четыре основных разновидности:

1. Непосредственная. Операнд «источник» является числом, причем его форма может быть десятичной, двоичной, восьмеричной или шестнадцатеричной:

```
Mov bx, 3; десятичная форма
Mov bx, 00001101b ; двоичная форма
Mov bx, 57o ; восьмеричная форма
Mov bx, 0bh ; шестнадцатеричная форма.
```

2. Регистровая. Операнды «источник» и «получатель» являются регистрами:

```
Mov ax, cx
```

3. Прямая. Если известен адрес операнда, располагающегося в памяти, можно использовать этот адрес. К примеру, строка

```
Mov ax, es:0001
```

помещает слово, находящееся в дополнительном сегментном регистре es (со смещением от начала сегмента 0001) в регистр ax. Кроме того, можно использовать директивы определения данных. Они дают программисту возможность использовать не адрес, а имя переменной. Тогда вышеуказанная строка приобретает следующий вид (если в сегменте, указанном в es, описана переменная word_1 размером в слово, то есть в 2 байта):

Mov ax, es:word_1.

Здесь ассемблер сам заменит текст «word_1» на соответствующий адрес.

4. Косвенная адресация. Из 16-разрядных регистров задействованы только bx, bp, si, di. 32-хразрядные используются практически все, за исключением сегментных и eip.

Для обозначения косвенной адресации применяются квадратные скобки [ ]. Суть данного вида адресации состоит в том, что внутри скобок содержится не необходимое нам значение, а ссылка на ячейку памяти, где оно хранится. Приведем примеры косвенной адресации:

Mov ax, [bx]; косвенная адресация с базированием

Mov ax, [di]; косвенная адресация с индексацией

Mov ax, [bp+si]; косвенная адресация с базированием и индексацией.

Помимо перечисленных видов адресации существуют и другие, на их основе, но о них будет упомянуто в других лабораторных работах.

Теперь перейдем к составлению программы и в качестве примера наберем несколько строк кода.

Программа 1:

```
mov al, 3 ; запись числа 3 в регистр ax
mov bx, ax ; копируем значение ax в bx
```

```
Mov dx, 300h; запись 300h в регистр портов ввода/вывода dx (инициализация
; порта ввода 300h)
In ax, dx ; водим данные из порта (300h) в регистр-аккумулятор ax
Add ax, bx ; производим сложение данных в ax и bx, помещая результат в ax
Mov dx, 301h ; инициализация порта вывода 301h
Out dx, ax ; вывод данных из аккумулятора (ax) в порт вывода 301h
Push ax ; заносим данные из ax в стек
Push bx ; заносим данные из bx в стек
Pop dx ; извлекаем значение из вершины стека в регистр dx
Pop bx ; извлекаем значение из вершины стека в bx
Std ; устанавливаем флаг направления DF в единицу
Cli ; сбрасываем флаг прерывания IF в ноль.
```

У Ассемблера есть еще одна особенность. По умолчанию нет никаких различий между строчными и прописными буквами. Поэтому можно использовать как те, так и эти – разницы не будет.

Набрав вышеприведенный участок кода, нажимаем кнопку запуска программы. Если при написании программы возникают какие-то ошибки, то компилятор выдаст их список с указанием строк, в которых они находятся. После исправления ошибок или же при их изначальном отсутствии появится следующая картина (рис. 6):

В правом верхнем окне показан собственно код программы. Подсветкой выделена та строка, которая будет сейчас выполняться.

В левом верхнем окне представлено довольно большое количество информации, но нам для начала требуются значения регистров. Отметим, что регистры общего назначения (ax, bx, cx, dx) разделены на старшую и младшую части размером в один байт. К примеру, регистр-аккумулятор ax делится на ah и al. Остальные представленные регистры не подразделяются на старшую и младшую части.

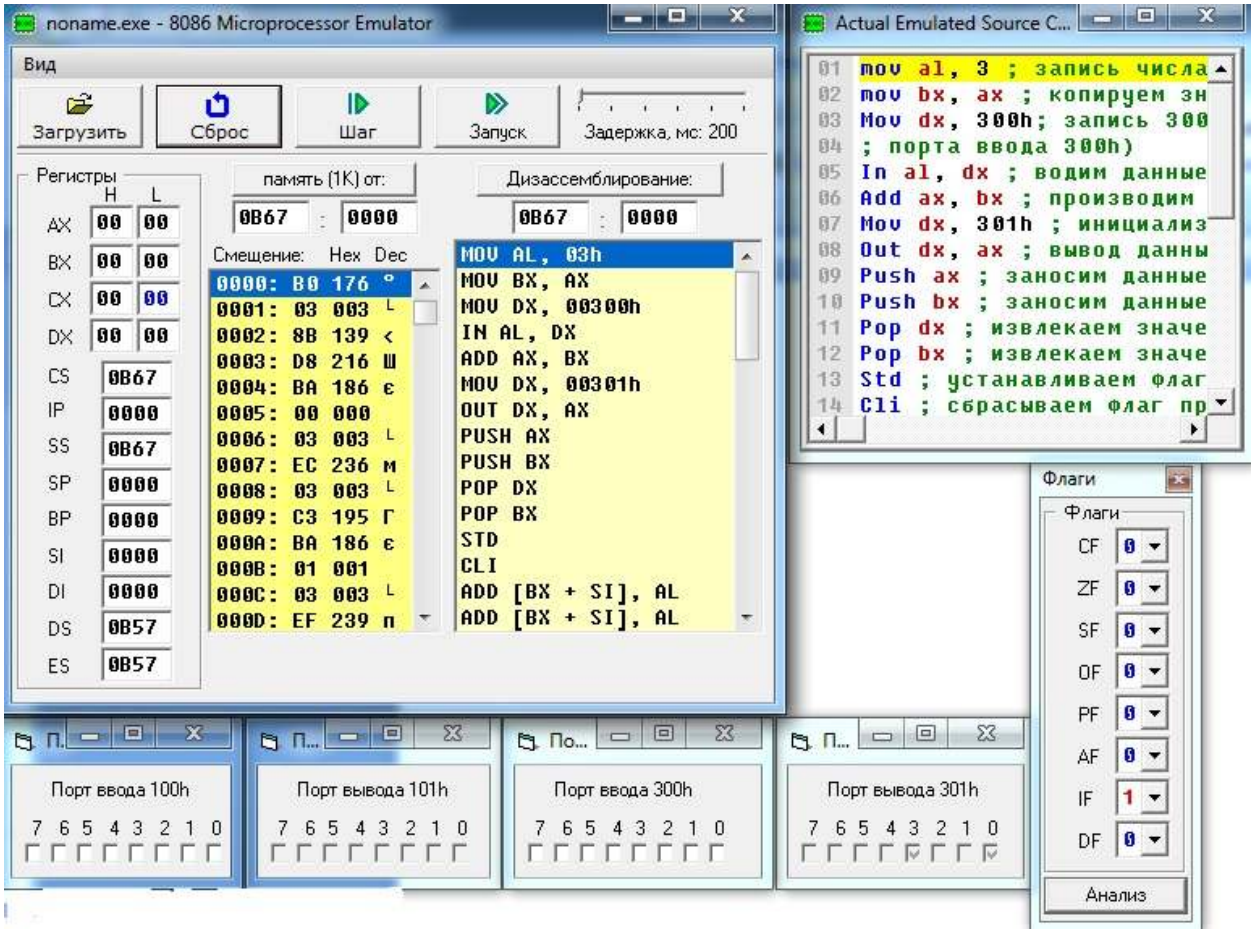

Рис. 6. Рабочие экраны эмулятора, показывающие процесс выполнения ассемблерных программ

В правом нижнем углу находятся четыре небольших окна, позволяющих работать с портами ввода/ вывода (100h, 300h – порты ввода. 101h, 301h – порты вывода). Как мы видим, в эмуляторе размер порта ограничен 1 байтом (8 битами). Для обучения этого более чем достаточно. Каждый квадратик соответствует биту. К примеру, если мы отмечаем галочкой квадратики под цифрами 1 и 2 в порту ввода 300h, то тем самым устанавливаем число в двоичном формате вида 00000110b. В десятичной системе это соответствует числу 6.

Следует учитывать, что мы не можем вручную установить значения битов в портах вывода. Внешне это отображается тем, что квадратики битов как бы залиты серым цветом. Значения в них меняются лишь после

инициализации портов вывода и срабатывания команд вывода данных в порты.

Наконец, в нижнем правом углу расположен регистр флагов. Как известно, флаги принимают значения 0 и 1 (истина или ложь). Как мы видим, по умолчанию установлен в 1 только флаг прерывания IF (разрешает использование прерываний).

Для того чтобы выполнить первую строчку кода, необходимо нажать кнопку «Шаг». После этого она выполнится, указатель строки перейдет на следующую позицию, а произошедшие изменения значений регистров будут выделены синим цветом (рис. 7).

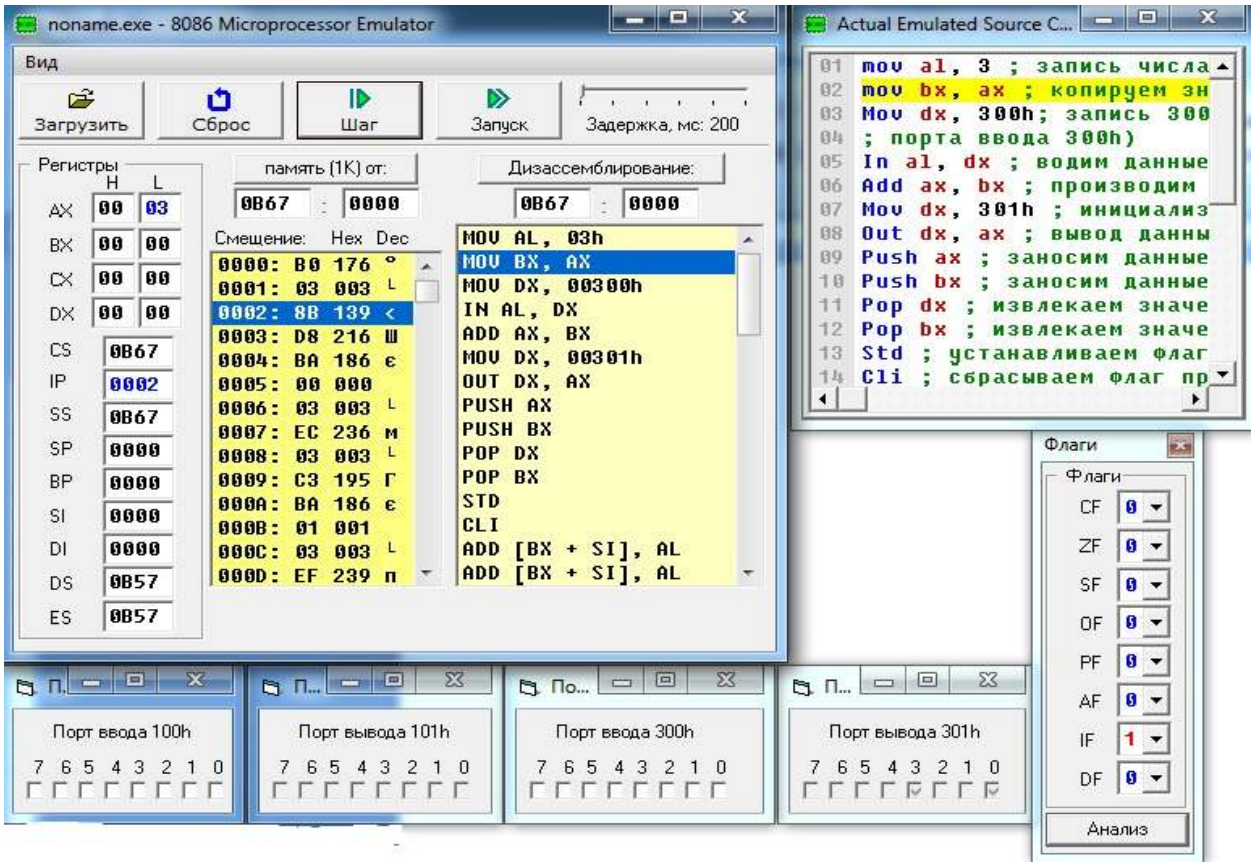

Рис. 7. Рабочие экраны эмулятора после выполнения первой строки кода

Выполняем программу дальше. При этом ввод данных в порт ввода 300h необходимо осуществить до выполнения строки «In ax, dx». В противном случае данный ввод не произведет никакого эффекта.

Выполнение строки «Add ax, bx», производящей сложение данных в ax и bx и помещающей результат в ax, в нашем случае приводит к еще одному эффекту (рис. 8).

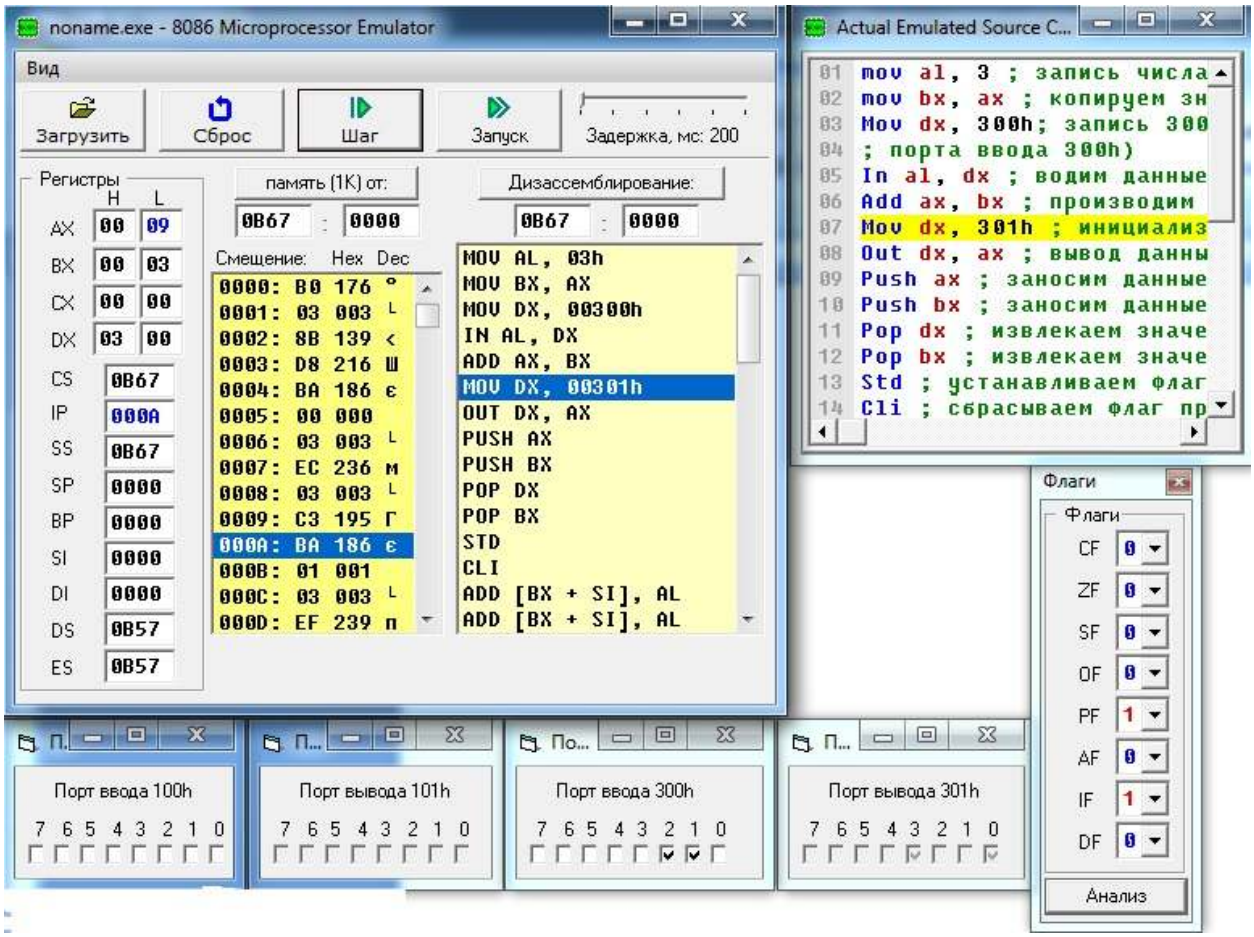

Рис. 8. Рабочие экраны эмулятора, показывающие процесс работы с портами ввода и изменение регистра флагов

Флаг четности PF установился в единицу. Это подтверждает то факт, что флаги меняются не сами по себе, а лишь в результате проводимых над данными операций или специальных команд. Сейчас сработал первый вариант.

Выполняем очередные строки программы вплоть до «Push ax». Поскольку эта команда будет работать со специальной областью памяти, называемой стеком, нам необходимо разобраться с принципами работы этого стека.

Стек – память, работающая по принципу LIFO (Last In – First Out) или «по принципу пистолетного магазина». То, что помещается туда первым, извлекается последним. Нарушение этого порядка не представляется возможным. Именно поэтому пользоваться данной специфической памятью нужно с большой осторожностью, хотя она и является крайне важной в разнообразных ситуациях.

Сначала в стеке нет интересующих нас данных. Затем там появляется число, скопированное из регистра ax, потом скопированное из регистра bx. Последнее смещает занесенное в стек ранее вглубь «магазина», как говорилось выше.

Затем, используя команду pop, мы извлекаем из стека верхнее значение и записываем его в указанный операнд-получатель. Сначала это регистр dx, а потом bx. В результате получаем картину, показанную на рисунке 9.

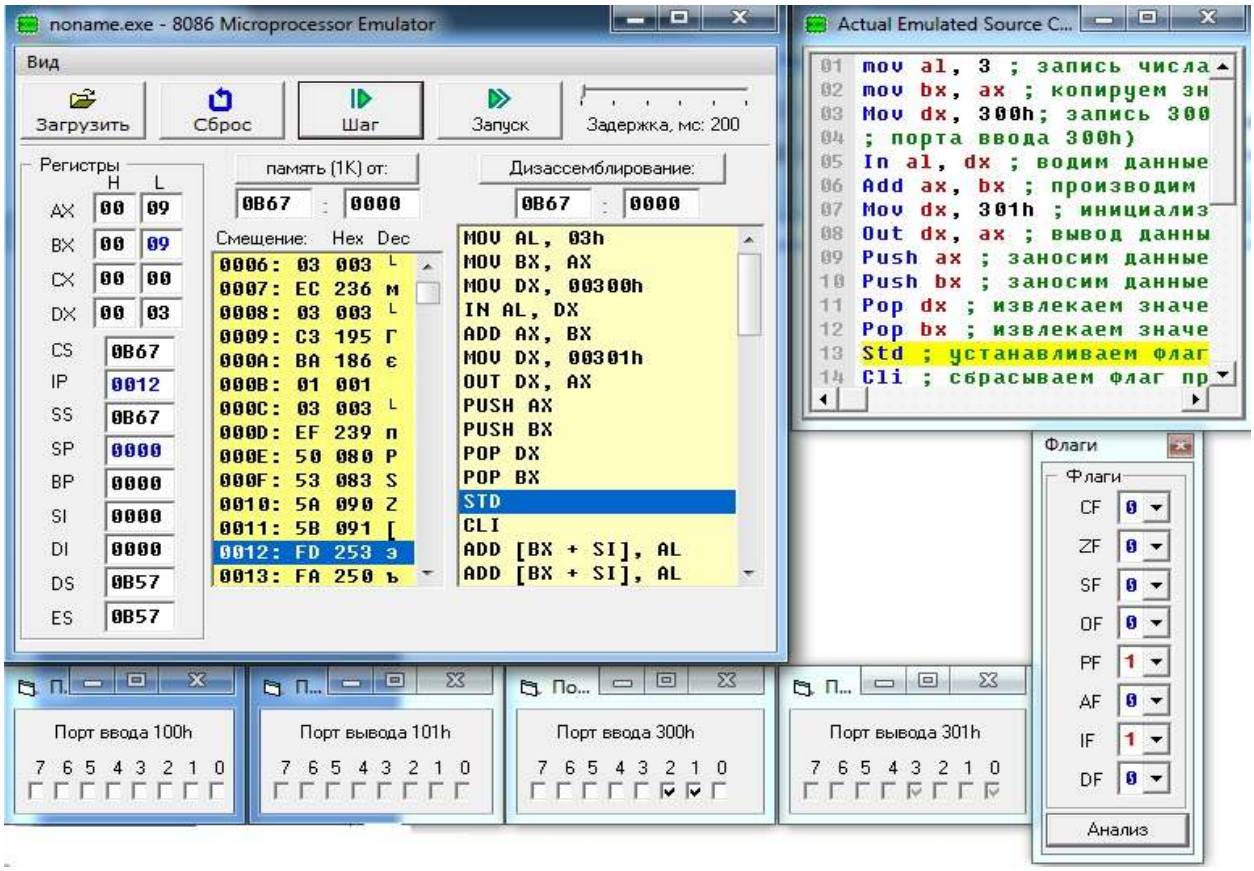

Рис. 9. Рабочие экраны эмулятора, показывающие процесс выполнения программы вплоть до работы с портом вывода информации

Последние две команды в данной программе предназначены для управления флагами. Std устанавливает флаг направления DF в единицу, а Cli сбрасывает флаг прерывания IF в ноль.

Условимся, что в этой и последующих лабораторных работах результаты по умолчанию будем заносить в таблицу следующего вида (рис. 10):

Кодировка	Регистры общего назначения								Регистр флагов	Порт ввода 300h/ (вывода 301h)
	AX		BX		CX		DX			
	AH	AL	BH	BL	CH	CL	DH	DL		

Рис. 10. Основная форма для оформления программ, входящих в лабораторные работы

При разборе работы небольшого участка кода нам становятся понятны азы работы языка низкого уровня Ассемблер. Однако для более полного знакомства с ним необходимо разобрать на этом же эмуляторе еще несколько фрагментов кода. Они позволят нам подробно изучить функционирование основных команд ассемблера. Только после этого мы можем перейти к написанию простых, но все же полноценных программ, а в дальнейшем перейти к работе не с эмулятором, а с нормальными оболочками языка Ассемблер, ориентированными на работу в средах DOS и Windows.

Программа 2:

```
Stc; устанавливаем в 1 флаг переноса CF
Pushf; заносим значения регистра флагов в стек
Cli; запрещаем прерывания (сбрасываем IF в 0)
Clc; сбрасываем в 0 флаг переноса CF
Popf; восстанавливаем из стека значение в регистр флагов
```

```
Mov ax, 2643h;
Mov bx, 0300h;
Xchg ax, bx; меняем местами значения операндов.
```

3. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

1. Изучить теоретическую часть методического пособия.

2. Открыть эмулятор Ассемблера и ввести в него коды программ 1 и 2, приведенных в пособии.

3. Пошагово выполняя программы, занести получаемые данные в таблицу (рис. 10).

4. Оформить протокол отчета, в который должны войти цель работы, коды программ и таблицы с полученными результатами.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Расскажите об известных вам регистрах процессора.

2. Регистр флагов, его структура.

3. Перечислите известные способы адресации и приведите примеры.

4. Стек, его особенности.

5. Порты ввода/вывода.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Зубков С.В. Assembler для DOS, Windows и UNIX. – М.: ДМК, 2006г. – 608 с.

2. Абель П. Ассемблер. Язык и программирование для IBM PC. – М.: Век+, 2009г. – 736 с.