# Лабораторная работа №1

Языки системного программирования

В данной лабораторной работе вы найдёте примеры простейших программ на MASM, а затем на основе имеющегося материала напишете свою программу. В теоретической части вы встретите вопросы, ответы на которые вы должны найти сами. К защите лабораторной работы вы должны быть готовы ответить на любой из них (разумеется, не ограничиваясь только ими).

## 1 Теоретическая справка

### 1.1 Кратко об устройстве процессора 80286

Некоторые регистры, которые можно использовать в арифметических операциях, называются регистры общего назначения. Их 8 штук, и все они имеют размер 16 бит. Хотя они и универсальны, у них также есть и особое назначение — в некоторых командах они используются неявно, без указания их имени. Из этого проистекают их названия.

- AX — аккумулятор, обычно используется в арифметических действиях;
- BX — база для базовой адресации;
- CX — счетчик цикла;
- DX — данные, например, прерывания DOS часто получают данные именно из DX;
- SI — Source Index, адрес ячейки-отправителя в командах работы со строками (MOVSB, ...);
- DI — Destination Index, адрес ячейки-получателя в командах работы со строками (MOVSB, ...);
- SP — хранит адрес последнего занесённого в стек элемента;
- BP — база для базовой адресации в случае, когда используются стековые фреймы. Хранит адрес начала текущего стекового фрейма (с этой темой вы подробно познакомитесь позднее).

К младшим и старшим 8 битам первых четырёх регистров можно обращаться отдельно, используя имена AL, AH и т.п. Например, если DX=0103, то DH = 01, DL = 03.

Помимо этого существуют особые 16-битные регистры, с которыми вы также столкнётесь:

- IP — счётчик команд;
- Flags — в своих битах хранит флаги процессора.
- CS, DS, SS — сегментные регистры, хранят адрес начала сегмента кода, данных и стека текущей программы. Линейный 20-разрядный адрес вычисляется умножением содержания соответствующего сегментного регистра на 16 и прибавлением смещения. Например, адрес следующей выполненной команды в форме сегмент:смещение записывается, как CS:IP, а считается, как CS * 16 + IP. Аналогично, вершина стека находится по адресу SS:SP.

## 1.2 Стек

Стек это структура данных типа Last In — First Out, которая ведёт себя подобно стопке тарелок. Новый элемент с помощью команды "pop" помещается на вершину стека, а с помощью команды "push" достаётся последний помещённый в стек элемент.

В наших компьютерах память линейно адресуема, это значит, что никакой стековой памяти у нас нет. Однако стек эмулируется с помощью машинных команд push, pop и пары регистров SS:SP. SS:SP хранит адрес вершины стека, а команды push и pop работают по следующим алгоритмам:

- push
  1. SP уменьшается на 2;
  2. По адресу из регистра SP кладётся значение (содержимое регистра или число, в зависимости от параметра команды).
- pop
  1. В регистр, указанный, в качестве параметра, кладётся значение, взятое по адресу из SS:SP;
  2. SP увеличивается на 2.

Из этого есть несколько важных выводов:

- Не бывает ситуации, когда "в стеке ничего нет тем более, если "мы туда ничего не положили". При выполнении, команда pop всегда будет следовать вышеуказанному алгоритму и выдавать соответствующий результат;
- Стек растёт вверх (к младшим адресам);
- В стек на 80286 можно положить или вынуть оттуда только одно слово (2 байта).

## 1.3 Hello, world!

По традиции начнём изучение языка ассемблера с программы "Hello, world!".

```
.model small
.stack 128

.data
mess db 'Hello world!$'

.code

main:

mov ax,@data
mov ds,ax

lea dx,mess
mov ah,09h
int 21h

mov ax,4c00h
int 21h

end main
```

## 1.4 Комментарий

**.model** small

Выбираем модель памяти small, что означает один сегмент кода и один сегмент данных.

**Вопрос 1.** *Что такое модель памяти?*

**Вопрос 2.** *Какие есть еще модели памяти?*

**.stack** 128

Выделяем 128 байт под стек. Это значит, что нам гарантировано, что если мы положим в стек не более 128 байт, то программа будет корректно работать.

**Вопрос 3.** *Что произойдёт, если размер стека не указать явно?*

**Вопрос 4.** *Может ли программа работать без стека?*

**.data**
mess **db** 'Hello world!$'

Объявляем начало сегмента данных и метку mess. Метка в ассемблере – имя для определённого адреса, так что в дальнейшем каждый раз при использовании имени mess вместо него будет подставляться смещение относительно начала сегмента данных.

.code

main:

Начало сегмента кода и метка main. В ассемблере можно произвольно смешивать описания кода и данных, каждый раз указывая начало соответствующего сегмента. Метка main это такой же адрес, впоследствии она будет нам важна – см. комментарий к последней строчке.

**mov ax**,@data
**mov ds**,**ax**

Нам необходимо настроить регистр DS, потому что иначе все смещения в сегменте данных будут отсчитываться от неправильного начала сегмента. В начале работы программы в DS лежит мусор. Вместо @data компилятор подставит корректный адрес начала сегмента данных, т.е. число. Например, команда:

**lea dx**,mess

использует адрес DS:mess (в формате адреса сегмент:смещение), поэтому для неё критично, чтобы значение DS было корректным. Напрямую переслать значение в DS нельзя, это не предусмотренно процессором, поэтому мы пересылаем его через регистр общего назначения.

Следующая команда загружает в DX смещение mess от начала сегмента данных.

**lea dx**,mess

**Вопрос 5.** *В чем разница между командами:*

***lea dx**,mess*
***mov dx**, mess*

*? Что будет в регистре DX после выполнения второй команды?*

**mov ah**,09h
**int** 21h

С помощью этих двух команд мы перемещаем в регистр AH значение 09h (h — суффикс шестнадцатиричной системы счисления) и вызываем вручную прерывание с номером 21h.

Аналогичная операция происходит в следующих двух строчках.

Директива end означает конец ассемблирования файла. Всё, что идёт после неё, игнорируется. Кроме того, её параметр указывает точку входа в программу – адрес, с которого она должна начать своё выполнение.

**end** main

**Вопрос 6.** *Что такое директива и чем она отличается от команды?*

**Вопрос 7.** *Объясните, что такое системный вызов и чем он отличается от обычной процедуры?*

Как вы видите, прерывания можно вызвать вручную с помощью команды int. Здесь они используются чтобы обратиться к системному вызову операционной системы DOS. Механизм системных вызовов DOS прост:

- В регистр AH заносится номер системного вызова;
- В другие регистры, возможно, заносятся параметры (зависит от системного вызова);
- Выполняется инструкция int 21h, вызывающая прерывание номер 21h, закреплённое за операционной системой DOS;
- Готово, некоторые системные вызовы возвращают в регистрах различные значения.

**Вопрос 8.** *Объясните, что такое системный вызов и чем он отличается от обычной процедуры?*

**Вопрос 9.** *В чем отличие инструкций call и int? Что такое call near и call far?*

Мы столкнулись уже с двумя системными вызовами:

- AH = 09h – вывод строки. Параметры: DS:DX = адрес начала строки.
- AH = 4ch – завершить выполнение программы и передать управление DOS.

  Параметры: AL = код возврата.

**Вопрос 10.** *Что такое код возврата?*

Рассмотрим также вызовы для вывода и считывания символов из консоли.

- AH = 01h – считывание символа из стандартного ввода, дублируя его в стандартный вывод.

  Возвращает: AL = считанный символ.
- AH = 02h – вывести символ на стандартный вывод.

  Параметры: DL = символ для вывода.

  Возвращает: AL = Последний выведенный символ.

Например, такая программа при запуске дублирует каждый введённый символ.

```
.model small
.code
start:
mov ah, 01h
int 21h
mov dl, al
mov ah, 02h
int 21h
jmp start
end start
```

**Вопрос 11.** *В каком случае после такого системного вызова AL не будет равен DL? (Подсказка: ищите полное описание системного вызова.)*

# 2 Задание

Ваша задача – написать калькулятор, работающий со стеком. Он должен реагировать на нажатия клавиш пользователем следующим образом:

1. Ввод числа от 0 до 9 — число кладётся на вершину стека, счётчик элементов в стеке увеличивается;
2. Ввод символа +/-

- если в стеке есть как минимум два числа, то последние два числа вынимаются оттуда, и результат их сложения/вычитания помещается в стек. Затем счётчик элементов в стеке изменяется так, чтобы отражать их реальное число. Затем вершина стека выводится на экран;
- если в стеке слишком мало чисел, то программа выдаёт на экран строчку "error"и игнорирует этот символ.

3. Ввод символа 'p' — вывод на экран числа, находящегося в вершине стека, или ошибка, если стек пуст. Для этого вы можете пользоваться процедурой вывода содержимого регистра AX на экран, которая находится в приложении.

4. Ввод символа 'q' — выход из программы. Желательно при этом очистить стек.

Несколько советов:

- Обратите внимание на то, что для того, чтобы очистить ваш стек, совершенно необязательно вынимать оттуда каждое лежащее там число. Такие решения неразумны и неэффективны, и не будут приниматься!
- В ассемблере можно писать символы в кавычках подразумевая их ASCII-коды.

## 2.1 Состав работы

- Отчёт
- Листинг
- Таблица трассировки
- Блок-схема алгоритма

# 3 Приложение

Процедура для вывода регистра AX в стандартный вывод. Пример использования — call ax_out.

```
;-------------------------------------
ax_out  proc  uses dx cx
; Outputs AX
;-------------------------------------

.code
push ax

mov ah,02h
mov cx, 12d

mainloop:

        pop dx
        push dx
        shr dx,cl
        and dl, 0Fh
        cmp dl, 10d
        jge add_37h
        add dl, 30h

        jmp print_4

        add_37h:
                add dl, 37h

        print_4:

                int 21h
        sub cx,3

loop mainloop

pop dx
push dx

and dl, 0fh
cmp dl, 10d
jge add_37h_last
add dl, 30h

jmp print_last

add_37h_last:
        add dl, 37h

print_last:
                int 21h

pop ax
ret
```