Соединила лекции и данные из Репы Леши

1. Регистры общего назначения.

Регистры — специальные ячейки памяти, находящиеся физически внутри процессора, доступ к которым осуществляется не по адресам, а по именам. Поэтому, работают очень быстро.

Существуют регистры, которые могут использоваться без ограничений, для любых целей — **регистры общего назначения.**

В 8086 регистры 16 битные.

При использовании регистров общего назначения, можно обратится к каждым 8 битам (байту) по-отдельности, используя вместо *X - *H или *L

Верхние 8 бит (1 байт) — AH, BH, CH, DH

Нижние 8 бит (1 байт) — AL, BL, CL, DL

- **AX** часто используется для хранения результата действий, выполняемых над двумя операндами. Например, используется при MUL и DIV (умножение и деление) (AX = AL * ЧИСЛО - при mul если число == байт, AL = AX / ЧИСЛО, при div если число == байт)

- **BX** - базовый регистр в вычислениях адреса, часто указывает на начальный адрес (называемый базой) структуры в памяти;

- **CX** - счетчик циклов, определяет количество повторов некоторой операции;

- **DX** - определение адреса ввода/вывода, так же может содержать данные, передаваемые для обработки в подпрограммы.

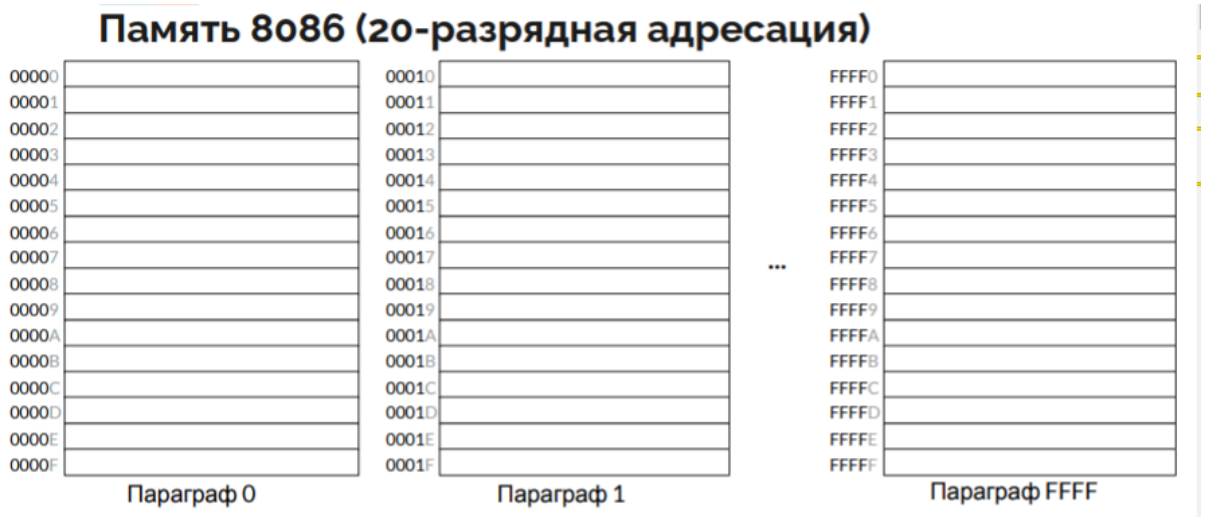
- **SI (индекс источника) и DI (индекс приемника) -** Ещё есть два этих регистра, они называются индексными, то есть используются для индексации в массивах / матрицах и т.д. (другие регистры (кроме BX и BP) не будут там работать (на 8086)).
    - Могут использоваться в большинстве команд, как регистры общего назначения.
    - В этих регистрах нельзя обратиться к каждому из байтов по-отдельности

2. Сегментные регистры. Адресация в реальном режиме.

- Сегмент кода - регистр CS. Командой MOV изменить невозможно, меняется автоматически по мере выполнения команд.
- Сегмент данных. Основной регистр - DS, при необходимости дополнительных сегментов данных задействуются ES, FS, GS.
- Сегмент стека - регистр SS

Мы знаем что регистр IP "указывает" на следующую команду, мы также знаем что регистры у нас размером в 16 бит, таким образом, используя один регистр программист имеет доступ только к 2^16 адресам, это примерно 64 кБ. Это достаточно мало, поэтому адресация в 8086 по 2^20 адресам, то есть примерно 1 МБ памяти. Для этого используют **адрес начала сегмента и смещение**. Сегментные регистры как раз хранят адрес. Реальный адрес высчитывается так: сегментная часть × 16 + смещение. (умножение на 16 = сдвиг на четыре бита влево)

При такой адресации адреса 0400h:0001h и 0000h:4001h будут ссылаться на одну и ту же ячейку памяти, так как 400h × 16 + 1 = 0 × 16 + 4001h.



3. Понятие сегментной части адреса и смещения.

[SEG]:[OFFSET] => физический адрес:

1. SEG необходимо побитово сдвинуть на 4 разряда влево (или умножить на 16, что тождественно)
2. К результату прибавить OFFSET

5678h:1234h =>

```
  56780
+ 1234
  579B4
```

4. Регистры работы со стеком.

Стек - структура данных, работающая по принципу LIFO (last in, first out) - последним пришёл, первым вышел.

Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний.

SP - указатель на вершину стека, **BP** - указатель на начало стека. BP используется в подпрограмме для сохранения "начального" значения SP, адресации параметров и локальных переменных.

В x86 стек ***растет вниз*, в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента.**

Команды непосредственной работы со стеком

- **PUSH <источник>**
  - Помещает данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- **POP <приемник>**
  - Считывает данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- **PUSHA**
  - Помещает в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.

- **POPA**
    - Загружает регистры из стека (SP игнорируется).

Вызов процедуры и возврат из процедуры

- **CALL <операнд>**
    - Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу либо IP либо CS:IP, в зависимости от размера аргумента. Передает управление на значение аргумента.
- **RET/RETN/RETF <число>**
    - Загружает из стека адрес возврата, увеличивает SP. Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров.

5. Структурная программа. Сегменты.

Любая программа состоит из сегментов

Виды сегментов:

- Сегмент кода
- Сегмент данных
- Сегмент стека

Описание сегмента в исходном коде:

имя SEGMENT READONLY *выравнивание тип разряд 'класс'*

…

имя ENDS

- Выравнивание - расположение начала сегмента с адреса, кратного какому-либо значению. Варианты:
    - BYTE,
    - WORD (2 байта),

    - DWORD (4 байта),
    - PARA (16 байт, по умолчанию),
- PAGE (256 байт).
- Тип:
    - PUBLIC (сегменты с одним именем объединятся в один) ----- заставляет компоновщик соединить все сегменты с одинаковым именем. Новый объединенный сегмент будет целым и непрерывным. Все адреса (смещения) объектов, а это могут быть, в зависимости от типа сегмента, команды или данные, будут вычисляться относительно начала этого нового сегмента;
    - STACK (для стека) ----- определение сегмента стека. Заставляет компоновщик соединить все одноименные сегменты и вычислять адреса в этих сегментах относительно регистра SS. Комбинированный тип STACK (стек) аналогичен комбинированному типу PUBLIC, за исключением того, что регистр SS является стандартным сегментным регистром для сегментов стека. Регистр SP устанавливается на конец объединенного сегмента стека. Если не указано ни одного сегмента стека, компоновщик выдаст предупреждение, что стековый сегмент не найден. Если сегмент стека создан, а комбинированный тип STACK не используется, программист должен явно загрузить в регистр SS адрес сегмента (подобно тому, как это делается для регистра DS);
    - COMMON (сегменты будут “наложены” друг на друга по одним и тем же адресам памяти); ------- располагает все сегменты с одним и тем же именем по одному адресу. Все сегменты с данным именем будут перекрываться и совместно использовать память. Размер полученного в результате сегмента будет равен размеру самого большого сегмента;
    - AT <начало> - расположение по фиксированному физическому адресу, параметр - сегментная часть этого адреса ----- располагает сегмент по абсолютному адресу параграфа (параграф — объем памяти, кратный 16, поэтому последняя шестнадцатеричная цифра адреса параграфа равна 0). Абсолютный адрес параграфа задается выражением xxxx. Компоновщик располагает сегмент по заданному адресу памяти (это можно использовать, например, для доступа к видеопамяти или области ПЗУ), учитывая атрибут комбинирования. Физически это означает, что сегмент при загрузке в память будет расположен, начиная с этого абсолютного адреса параграфа, но для доступа к нему в соответствующий сегментный регистр должно быть загружено заданное в атрибуте значение. Все метки и адреса в определенном таким образом сегменте отсчитываются относительно заданного абсолютного адреса;

```
DS2 SEGMENT AT 0b800h
    CA LABEL byte
    ORG 80 * 2 * 2 + 2 * 2
    SYMB LABEL word
DS2 ENDS
```

  - PRIVATE - вариант по умолчанию. ----- сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля.
- Класс - метка, позволяющая объединить сегменты (расположить в памяти друг за другом).

пример

```
; Сегмент данных
DATAS SEGMENT PARA PUBLIC 'DATA'
    LIMIT2 DW 1
    LIMIT16 DW 15
    SIGNBINNUMBER DB 16 DUP('0'), '$'
    UNSIGNHEXNUMBER DB 4 DUP('0'), '$'
    SIGNFORHEX DB ' '
DATAS ENDS
```

Директива ASSUME

ASSUME ***регистр*** **:** ***имя сегмента***

- Не является командой
- Нужна для контроля компилятором правильности обращения к переменным

Модель памяти

.model модель, язык, модификатор

- TINY - один сегмент на всё
- SMALL - код в одном сегменте, данные и стек - в другом
- COMPACT - допустимо несколько сегментов данных
- MEDIUM - код в нескольких сегментах, данные - в одном
- LARGE, HUGE
- *Язык* - C, PASCAL, BASIC, SYSCALL, STDCALL. Для связывания с ЯВУ и вызова подпрограмм.
- *Модификатор* - NEARSTACK/FARSTACK

Определение модели позволяет использовать сокращённые формы директив определения сегментов.

Конец программы и точка входа

END start

- start - имя метки, объявленной в сегменте кода и указывающее на команду, с которой начнётся исполнение программы.
- Если в программе несколько модулей, только один может содержать начальный адрес.

6. Прерывание 21h. Примеры ввода-вывода.

- Аналог системного вызова в современных ОС.
- Используется наподобие вызова подпрограммы.
- Номер функции передается через AH.

Еще одним важным случаем, когда нам требуется прерывание DOS - завершение программы. Чтобы ассемблер перестал читать подряд строки кода нам требуется положить в ah код DOS завершения программы (04Ch) и вызвать прерывание. Код завершения программы (ошибка или нет) кладется и берется из al.

функция	назначение	вход	выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом $	-
01	Считать символ из stdin с эхом	-	AL – ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL=FF	AL – ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL – ASCII-код символа

08	Считать символ без эха	-	AL – ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры - AL=0, если клавиша не была нажата, и FF, если была		
OCh	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	-

```
    MOV AH,4Ch
    INT 21h
```

завершение, код можно поместить в регистр AL

```
; Ввода символа
INPUTSYM:
    MOV AH, 1
    INT 21h
    RET
```

```
; Вывод сообщения
OUTPUTMSG:
    MOV AH, 9
    INT 21h
    RET
```

7. Стек. Назначение. Примеры использования.

Стек работает по правилу LIFO / FILO (последним пришёл, первым вышел)

Сегмент стека — область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний.

Используется для временного хранения переменных, передачи параметров для подпрограмм, адрес возврата при вызове процедур и прерываний.

Регистр SP — указывает на вершину стека

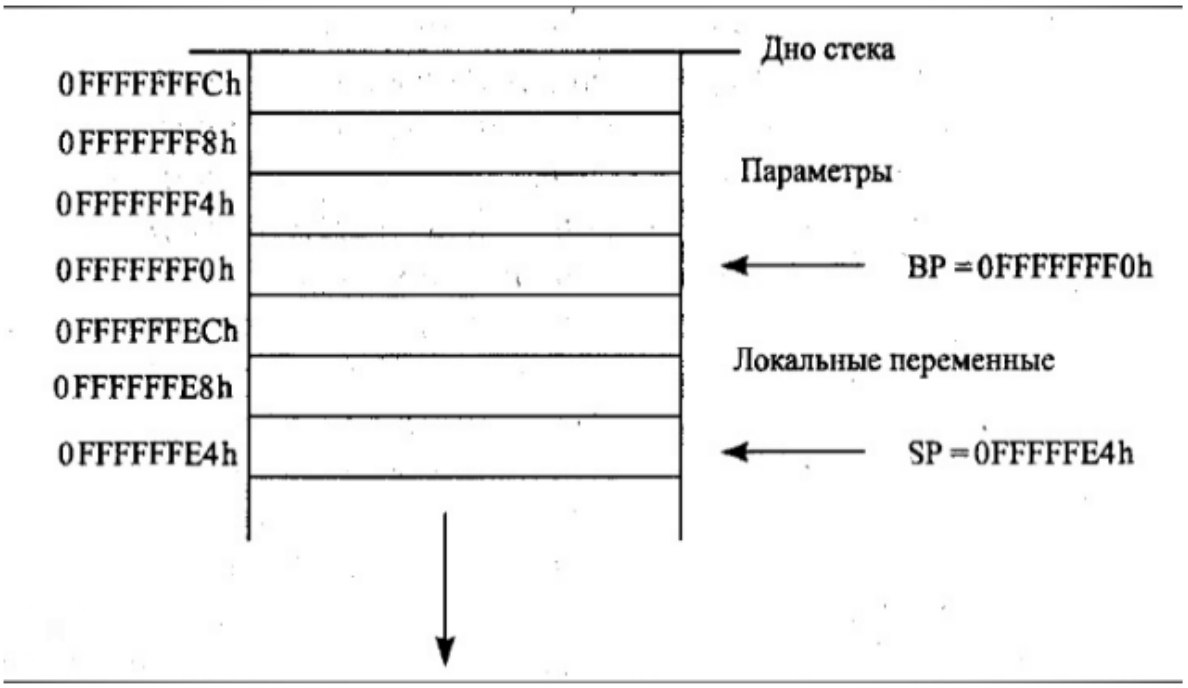
В x86 стек "растет вниз", в сторону уменьшения адресов (от максимально возможно адреса). При запуске программы SP указывает на конец сегмента.

BP (Base Pointer)

Используется в подпрограмме для сохранения "начального" значения SP.

Так же, используется для адресации параметров и локальных переменых.

При вызове подпрограммы параметры кладут на стек, а в BP кладут текущее значение SP. Если программа использует стек для хранения локальных переменных, SP изменится и таким образом можно будет считывать переменные напрямую из стека (их смещения запишутся как BP + номер параметра)



Команды непосредственной работы со стеком

- **PUSH <источник>**
    - Помещает данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- **POP <приемник>**
    - Считывает данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- **PUSHA**
    - Помещает в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- **POPA**

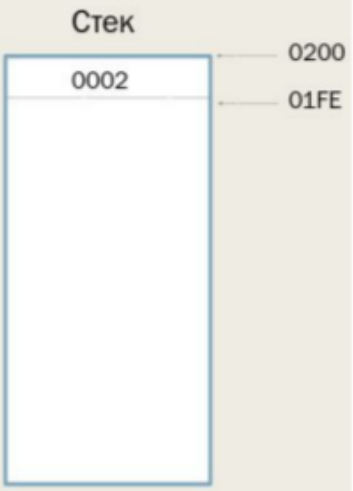
- Загружает регистры из стека (SP игнорируется).

CALL и RET

- CALL <операнд> — передает управление на адрес <операнд>
  - Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- RET <число> — загружает из стека адерс возврата, увеличивая SP.
  - Если указать операнд, то можно очистить стек для очистки стека от параметров (<число> будет прибавлено к SP)

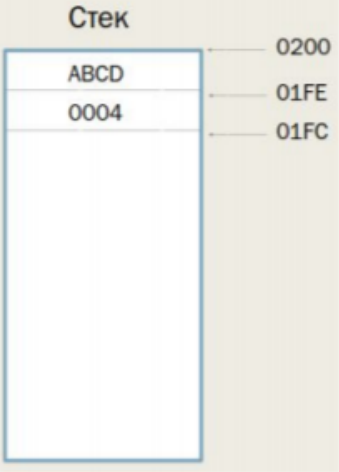
Пример вызова подпрограммы №1

```
                    0. SP = 0200
0000: CALL P1       1. SP = 01FE
0002: MOV BX, AX
…
P1:
0123: MOV AX, 5
0125: RET           2. SP = 0200
```

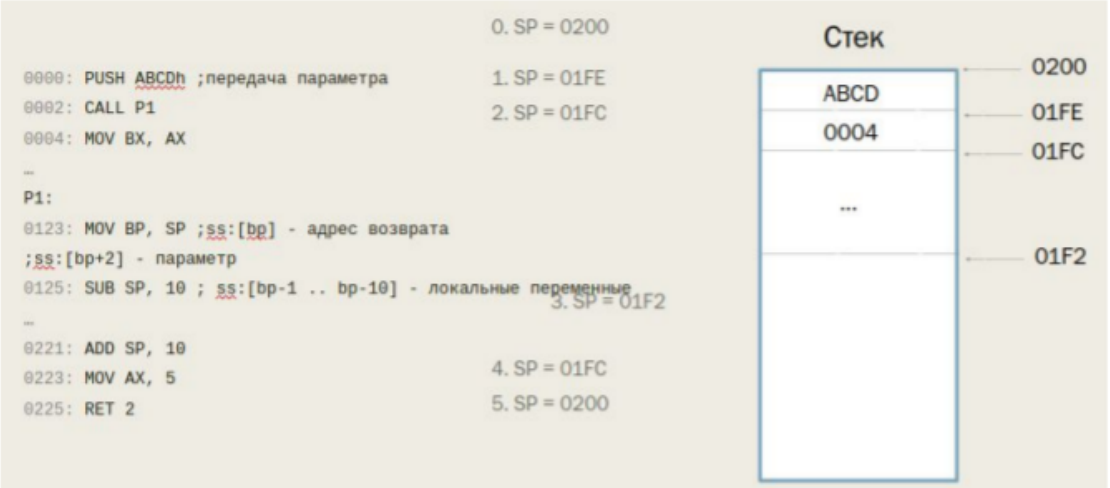


Пример вызова подпрограммы №2

```
                                    0. SP = 0200
0000: PUSH ABCDh ;передача параметра 1. SP = 01FE
0002: CALL P1                       2. SP = 01FC
0004: POP DX                        4. SP = 0200
0006: MOV BX, AX
…
P1:
0123: MOV BP, SP ;ss:[bp] - адрес возврата
;ss:[bp+2] - параметр
...
0223: MOV AX, 5
0225: RET                           3. SP = 01FE
```



Пример вызова подпрограммы №3



8. Регистры флагов.

Флаги выставляются при операциях, но не обязательно все сразу. Например INC и DEC не затрагивают флаг CF, в отличии от ADD и SUB.

Также есть команды рассчитанные на флаги, например CMP, которая выставляет флаги такие, как если бы произошло вычитание аргументов.

Как мы помним, регистры у нас размером в 16 бит.

Регистр FLAGS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CF	-	PF	-	AF	-	ZF	SF	TF	IF	DF	OF	IOPL		NT	-

- CF (carry flag) - флаг переноса
- PF (parity flag) - флаг чётности
- AF (auxiliary carry flag) - вспомогательный флаг переноса
- ZF (zero flag) - флаг нуля
- SF (sign flag) - флаг знака
- TF (trap flag) - флаг трассировки
- IF (interrupt enable flag) - флаг разрешения прерываний
- DF (direction flag) - флаг направления
- OF (overflow flag) - флаг переполнения
- IOPL (I/O privilege flag) - уровень приоритета ввода-вывода
- NT (nested task) - флаг вложенности задач

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CF	-	PF	-	AF	-	ZF	SF	TF	IF	DF	OF	IOPL		NT	-

зеленый DF - флаг управления
черные - флаги состояния
синие - системные флаги
CF TF IF DF - можно изменять командами

- **CF (carry flag)** - флаг переноса - устанавливается в 1, если результат предыдущей операции не уместился в приемник и произошел перенос или если требуется заем при вычитании. Иначе 0.
- **PF (parity flag)** - флаг четности - устанавливается в 1, если младший байт результата предыдущей операции содержит четное количество единиц.
- **AF (auxiliary carry flag)** - вспомогательный флаг переноса - устанавливается в 1, если в результате предыдущей операции произошел перенос из 3 в 4 или заем из 4 в 3 биты.
- **ZF (zero flag)** - флаг нуля - устанавливается в 1, если если результат предыдущей команды равен 0.
- **SF (sign flag)** - флаг знака - всегда равен старшему биту результата.
- **TF (trap flag)** - флаг трассировки - предусмотрен для работы отладчиков в пошаговом режиме. Если поставить в 1, после каждой команды будет происходить передача управления отладчику.
- **IF (interrupt enable flag)** - флаг разрешения прерываний - если 0 процессор перестает обрабатывать прерывания от внешних устройств.
- **DF (direction flag)** - флаг направления - контролирует поведение команд обработки строк. Если 0, строки обрабатываются слева направо, если 1 справа налево.
- **OF (overflowflag)** - флаг переполнения - устанавливается в 1, если результат предыдущей операции над числами со знаком выходит за допустимые для них пределы.
- **IOPL (I/O privilege level)** - уровень приоритета ввода-вывода - а это на 286, на не нужно пока.
- **NT (nested task)** - флаг вложенности задач - а это на 286, на не нужно пока.

9. Команды условной и безусловной передачи управления.

Условный переход - переход, происходящий при выполнении какого-то условия.

Безусловный переход - переход, не зависящий от чего-либо (совершаемый в любом случае).

- **Виды безусловных переходов**
  - JMP - оператор безусловного перехода.
  - Для короткого и ближнего переходов непосредственный операнд (число) прибавляется к IP. Регистры и переменные заменяют старое значение в IP (CS:IP).

Вид перехода	Дистанция перехода
short (короткий)	-128..+127 байт
near (ближний)	в том же сегменте (без изменения CS)
far (дальний)	в другой сегмент (со сменой CS)

- **Виды условных переходов**
  - ***Не зависящие от знака***

Команда	Описание	Состояние флагов для выполнения перехода
JO	Есть переполнение	OF = 1
JNO	Нет переполнения	OF = 0
JS	Есть знак	SF = 1
JNS	Нет знака	SF = 0
JE/JZ	Если равно/если ноль	ZF = 0
JNE/JNZ	Если не равно/если не ноль	ZF = 0
JP/JPE	Есть четность/четное	PF = 1
JNP/JPO	Нет четности/нечетное	PF = 0

JCXZ	CX = 0	

- ***Беззнаковые***

Команда	***Описание***	***Состояние флагов для выполнения перехода***	***Знаковый***
JB/JNAE/JC	***Если ниже/если не выше и не равно/если перенос***	***CF = 1***	***нет***
JNB/JAE/JNC	***Если не ниже/если выше и равно/если перенос***	***CF = 0***	***нет***
JBE/JNA	***Если ниже или равно/если не выше***	***CF = 1 или ZF = 1***	***нет***
JB/JNAE/JC	***Если ниже/если не выше и не равно/если перенос***	***CF = 1***	***нет***
JA/JNBE	***Если выше/если не ниже и не равно***	***CF = 0 и ZF = 0***	***нет***

- ***Знаковые***

Команда	**Описание**	**Состояние флагов для выполнения перехода**	**Знаковый**

JL/JNGE	Если меньше/если не больше и не равно	SF != OF	да
JGE/JNL	Если больше или равно/если не меньше	SF = OF	да
JLE/JNG	Если меньше или равно/если не больше	ZF = 1 или SF != OF	да
JG/JNLE	Если больше/если не меньше и не равно	ZF = 0 и SF = OF	да

- **Команды, выставляющие флаги и использующиеся при переходах к передаче управления**
  - **CMP <приемник>, <источник>**
    - Источник - число, регистр или переменная.
    - Приемник - регистр или переменная; не может быть переменной одновременно с источником.
    - Вычитает источник из приёмника, результат никуда не сохраняется, выставляются флаги CF, PF, AF, ZF, SF, OF.
    - *Возможно, если равны числа, то ZF == 1, если первое < второго, то CF == 1*
  - **TEST <приемник>, <источник>**
    - Аналог AND, но результат не сохраняется. Выставляются флаги SF, ZF, PF.

Инструкция AND выполняет логическое И между всеми битами двух операндов. Результат записывается в первый операнд. Синтаксис:

```
AND ЧИСЛО1, ЧИСЛО2
```

ЧИСЛО1 может быть одним из следующих:

- Область памяти (MEM)
- Регистр общего назначения (REG)

ЧИСЛО2 может быть одним из следующих:

- Область памяти (MEM)
- Регистр общего назначения (REG)
- Непосредственное значение (IMM)

( )

10. Организация многомодульных программ.

Как и на других языках программирования, программа на ассемблере может состоять из нескольких файлов - модулей. При компиляции (трансляции) каждый модуль превращается в объектный файл, далее при компоновке объектные файлы соединяются в единый исполняемый модуль.
Модули обычно состоят из описания сегментов будущей программы с помощью директивы SEGMENT.

Пример:

```
имя SEGMENT [READONLY] выравнивание тип разряд 'класс'
…
имя ENDS
```

Параметры:

- ● Выравнивание - расположение начала сегмента с адреса, кратного какому-либо значению. Варианты:
  - BYTE;
  - WORD (2 байта);
  - DWORD (4 байта);
  - PARA (16 байт, по умолчанию);
  - PAGE (256 байт).
- ● Тип:
  - PUBLIC (сегменты с одним именем объединятся в один);
  - STACK (для стека); COMMON (сегменты будут “наложены” друг на друга по одним и тем же адресам памяти);
  - AT <начало> - расположение по фиксированному физическому адресу, параметр - сегментная часть этого адреса;
  - PRIVATE - вариант по умолчанию.
- ● Класс - метка, позволяющая объединить сегменты (расположить в памяти друг за другом).

11. Подпрограммы. Объявления, вызов.

Описание подпрограммы

```
имя_подпрограммы PROC [NEAR | FAR] ; по умолчанию NEAR, если не указать
    ;тело подпрограммы;
    ret [кол-во используемых локальный переменных] ; ничего не указывается, если не использовались локальные переменные на стеке
имя_подпрограммы ENDP
```

Вызов подпрограммы

```
; вызов любой (в плане расстояния) подпрограммы

call имя_подпрограммы
```

CALL - вызов процедуры, RET - возврат из процедуры

- CALL
    - Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
    - Передаёт управление на значение аргумента.
- RET/RETN/RETF
    - Загружает из стека адрес возврата, увеличивает SP
    - Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров
    - Отличие RETN и RETF в том, что 1ая команда делает возврат при ближнем переходе, 2ая - при дальнем (различие в кол-ве байт, считываемых из стека при возврате). Если используется RET, то ассемблер сам выберет между RETN и RETF в зависимости от описание подпрограммы (процедуры).
- BP – base pointer
    - Используется в подпрограмме для сохранения "начального" значения SP
    - Адресация параметров
    - Адресация локальных переменных

12. Арифметические команды.

ADD и ADC

ADD <приемник>, <источник> — сложение. Не делает различий между знаковыми и беззнаковыми числами.

ADC <приемник>, <источник> — сложение с переносом. Складывает приёмник, источник и флаг CF.

SUB и SBB

SUB <приемник>, <источник> — вычитание. Не делает различий между знаковыми и беззнаковыми числами.

SBB <приемник>, <источник> — вычитание с займом. Вычитает из приёмника источник и дополнительно - флаг CF.

Флаг CF можно рассматривать как дополнительный бит у результата.

$$11111111_2 + 00000001_2 = (1)00000000_2 \text{ (флаг установлен)}$$

.

Можно использовать ADC и SBB для сложения вычитания и больших чисел, которые по частям храним в двух регистрах.

Пример: Сложим два 32-битных числа. Пусть одно из них хранится в паре регистров DX:AX (младшее двойное слово - DX, старшее AX). Другое в паре BX:CX

```
add ax, cx

adc dx, bx
```

Если при сложении двойных слов произошел перенос из старшего разряда, то это будет учтено командой adc.

Эти 4 команды (ADD, ADC, SUB, SBB) меняют флаги: CF, OF, SF, ZF, AF, PF

MUL и IMUL

MUL <источник> — выполняет умножение чисел без знака. <источник> не может быть число (нельзя: MUL 228). Умножает регистр AX (AL), на <источник>. Результат остается в AX, либо DX:AX, если не помещается в AX.

IMUL — умножение чисел со знаком.

1. IMUL <источник>. Работает так же, как и MUL
2. IMUL <приёмник>, <источник>. Умножает источник на приемник, результат в приемник.
3. IMUL <приёмник>, <источник1>, <источник2>. Умножает источник1 на источник2, результат в приёмник.

Флаги: OF, CF

DIV и IDIV

DIV <источник> — выполняет деление чисел без знака. <источник> не может быть число (нельзя: DIV 228). Делимое должно быть помещено в AX (или DX:AX, если делитель больше байта). В первом случае частное в AL, остаток в AH, во втором случае частное в AX, остаток в DX.

IDIV <источник> — деление чисел со знаком. Работает так же как и DIV. Округление в сторону нуля, знак остатка совпадает со знаком делимого.

INC, DEC, NOT

INC <приемник> — увеличивает примник на 1.

DEC <приемник> — уменьшает примник на 1.

Меняют флаги: OF, SF, ZF, AF, PF

NEG <применик> — меняет знак приемника.

13. Команды побитовых операций.

Операции над битами и байтами

- **BT <база>, <смещение>**
  - Считывает в CF значение бита из битовой строчки.
- **BTS <база>, <смещение>**
  - Устанавливает бит в 1.
- **BTR <база>, <смещение>**
  - Сбрасывает бит в 0.
- **BTC <база>, <смещение>**
  - Инвертирует бит.
- **BSF <приемник>, <смещение>**
  - Прямой поиск бита (от младшего разряда).
- **BSR <приемник>, <смещение>**
  - Обратный поиск бита (от старшего разряда).
- **SETcc <приемник>**
  - Выставляет приемник (1 байт) в 1 или 0 в зависимости от условия, аналогично *Jcc*.

    jcc метка Проверка состояния флагов в зависимости от кода операции (оно отражает проверяемое условие):

    - если проверяемое условие истинно, то перейти к ячейке, обозначенной операндом;
    - если проверяемое условие ложно, то передать управление следующей команде.

Логический, арифметический, циклический сдвиг

- **SAL (SHL)**
  - Арифметический сдвиг налево.
- **SHR**
  - Арифметический сдвиг направо, зануляет старший бит.

- **SAR**
    - Арифметический сдвиг направо, сохраняет знак.
- **ROR (ROL)**
    - Циклический сдивг вправо (влево).
- **RCR (RCL)**
    - Циклический сдвиг вправо (влево) через CF.

14. Команды работы со строками.

Строка-источник - `DS:SI`, строка-приёмник - `ES:DI`.

За один раз обрабатывается один байт (слово).

- `MOVS` / `MOVSB` / `MOVSW <приёмник>, <источник>` - копирование

  Алгоритм работы

  - выполнить копирование байта, слова или двойного слова из операнда источника в операнд приемник, при этом адреса элементов предварительно должны быть загружены:
    - адрес источника — в пару регистров ds:esi/si (ds по умолчанию, допускается замена сегмента);
    - адрес приемника — в пару регистров es:edi/di (замена сегмента не допускается);
  - в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:
    - если df=0, то увеличить содержимое этих регистров на длину структурного элемента последовательности;
    - если df=1, то уменьшить содержимое этих регистров на длину структурного элемента последовательности;
  - если есть префикс повторения, то выполнить определяемые им действия (см. команду rep).

- `CMPS` / `CMPSB` / `CMPSW <приёмник>, <источник>` - сравнение

  Алгоритм работы:

  - выполнить вычитание элементов (источник - приемник), адреса элементов предварительно должны быть загружены:
    - адрес источника — в пару регистров ds:esi/si;
    - адрес назначения — в пару регистров es:edi/di;
  - в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:
    - если df=0, то увеличить содержимое этих регистров на длину элемента последовательности;
    - если df=1, то уменьшить содержимое этих регистров на длину элемента последовательности;
  - в зависимости от результата вычитания установить флаги:
    - если очередные элементы цепочек не равны, то cf=1, zf=0;
    - если очередные элементы цепочек или цепочки в целом равны, то cf=0, zf=1;
  - при наличии префикса выполнить определяемые им действия (см. команды repe/repne).

- SCAS / SCASB / SCASW <приёмник> - сканирование (сравнение с AL/AX (в зависимости от размеров приемника)

  Алгоритм работы:

  - выполнить вычитание (элемент цепочки-(eax/ax/al)). Элемент цепочки локализуется парой es:edi/di. Замена сегмента es не допускается;
  - по результату вычитания установить флаги;
  - изменить значение регистра edi/di на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:
    - df=0 — величина положительная, то есть просмотр от начала цепочки к ее концу;
    - df=1 — величина отрицательная, то есть просмотр от конца цепочки к ее началу.

- LODS / LODSB / LODSW <источник> - чтение (в AL/AX)

  Алгоритм работы:

  - загрузить элемент из ячейки памяти, адресуемой парой ds:esi/si, в регистр al/ax/eax. Размер элемента определяется неявно (для команды lods) или явно в соответствии с применяемой командой (для команд lodsb, lodsw, lodsd);
  - изменить значение регистра si на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:
    - df=0 — значение положительное, то есть просмотр от начала цепочки к ее концу;
    - df=1 — значение отрицательное, то есть просмотр от конца цепочки к ее началу.

- STOS / STOSB / STOSW <приёмник> - запись (из AL/AX)

  Алгоритм работы:

  - записать элемент из регистра al/ax/eax в ячейку памяти, адресуемую парой es:di/edi. Размер элемента определяется неявно (для команды stos) или конкретной применяемой командой (для команд stosb, stosw, stosd);
  - изменить значение регистра di на величину, равную длине элемента цепочки. Знак этого изменения зависит от состояния флага df:
    - df=0 — увеличить, что означает просмотр от начала цепочки к ее концу;
    - df=1 — уменьшить, что означает просмотр от конца цепочки к ее началу.

- Префиксы: REP / REPE / REPZ / REPNE / REPNZ
- REP - повторить следующую строковую операцию
- REPE - повторить следующую строковую операцию, если равно
- REPZ - Повторить следующую строковую операцию, если нуль
- REPNE - повторить следующую строковую операцию, если не равно
- REPNZ - повторить следующую строковую операцию, если не нуль

Префиксы REP (F3h), REPE (F3h) и REPNE (F2h) применяются со строковыми операциями. Каждый префикс заставляет строковую команду, которая следует за ним, повторяться указанное в регистре счетчике (E)CX (в случае нашей

модели процессора 8086 - CX) количество раз или, кроме этого, (для префиксов REPE и REPNE) пока не встретится указанное условие во флаге ZF.

Пример использования: REP LODS AX

Мнемоники REPZ и REPNZ являются синонимами префиксов REPE и REPNE соответственно и имеют одинаковые с ними коды. Префиксы REP и REPE / REPZ также имеют одинаковый код F3h, конкретный тип префикса задается неявно той командой, перед которой он применен.

Все описываемые префиксы могут применяются только к одной строковой команде за один раз. Чтобы повторить блок команд, используется команда LOOP или другие циклические конструкции.

Затрагиваемые флаги: OF, DF, IF, TF, SF, ZF, AF, PF, CF

15. Прерывания. Обработка прерываний.

Прерывание означает временное прекращение основного процесса вычислений для выполнения некоторых запланированных или незапланированных действий, вызванных работой устройств или программы.

В зависимости от источника различают прерывания:

- **Аппаратные** (внешние) – реакция процессора на физический сигнал от некоторого устройства. Возникают в случайные моменты времени, а значит – асинхронные
- **Программные** (внутренние) – возникает в заранее запланированный момент времени — синхронные
- **Исключения** – **разновидность программных прерываний, реакция процессора на некоторую не стандартную ситуацию возникшую во время выполнения команды;**

Вектор прерываний – адрес процедуры обработки прерываний. Адреса размещаются в специальной области памяти доступной для всех подпрограмм. Вектор прерывания одержит 4 байта: старшее слово содержит сегментную составляющую адреса процедуры обработки исключения, младшее — смещение.

Вызов прерывания осуществляется с помощью директивы **INT номер_прерывания**.

00h – 1Fh – прерывания BIOS

20h – 3Fh – прерывания DOS

40h – 5Fh – зарезервировано

60h – 7Fh – прерывания пользователя

80h – FFh – прерывания Бейсика

Порядок выполнения INT:

1) В стек помещается содержимое регистра флагов FLAGS

2) Флаги трассировки и прерывания устанавливаются в нуль (TF = 0, IF = 0)

3) Вычисляется адрес соответствующего вектора прерываний – 4*номер_прерывания

4) Содержимое сегментного регистра CS помещается в стек

5) Содержимое второго слова вектора прерываний заносится в CS

6) Содержимое управляющего регистра IP заносится в стек

7) Первое слово вектора прерываний помещается в IP

Процедура используемая для обработки прерывания должна быть дальней (FAR). Возврат с прерывания осуществляется при помощи директивы **IRET**, которая восстанавливает CS:IP и FLAGS. Все параметры в процедуру и из нее передаются через регистры, какое – зависит от прерывания.

- Прерывание - особая ситуация, когда выполнение текущей программы приостанавливается и управление передаётся программе-обработчику возникшего прерывания.
- Виды прерываний:
  - аппаратные (асинхронные) - события от внешних устройств;
  - внутренние (синхронные) - события в самом процессоре, например, деление на ноль;
  - программные - вызванные командой INT

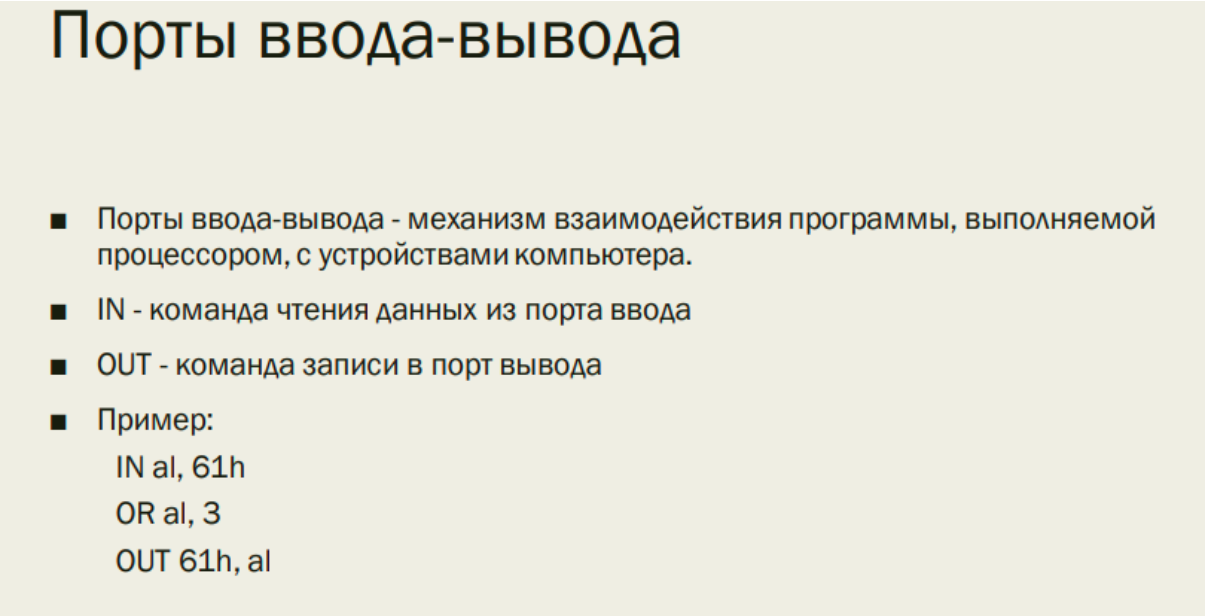
Прерывание DOS - вывод на экран в текстовом режиме

Функция	Назначение	Вход	Выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом $	-

Прерывание DOS - ввод с клавиатуры

Функция	Назначение	Вход	Выход
01	Считать символ из stdin с эхом	-	AL - ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL = FF	AL - ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL - ASCII-код символа
08	Считать символ без эха	-	AL - ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры	-	AL=0, если клавиша не была нажата, и FF, если была
0Ch	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	

16. Работа с портами ввода-вывода.



РК 2020 ГОДА

1. Вычислить физический адрес, соответствующий следующим сегментной части и смещению: 1A29h:37B4h
   Вычисляем по схеме
   1) seg побитого сдвигаем на 4 разряда влево / умножить на 16
   2) Прибавить offset
   ответ 1DA44h

   [SEG]:[OFFSET] => физический адрес:
   1. SEG необходимо побитово сдвинуть на 4 разряда влево (или умножить на 16, что тождественно)
   2. К результату прибавить OFFSET

   5678h:1234h =>
   56780
   +1234
   579B4

2. Какова разрядность регистра IP?
   разрядность всех регистров по 16
   ответ 16

3. Какая формулировка соответствует характеристикам короткого перехода?

Виды переходов (передачи управления)

Короткий (short) - в пределах адресов -128..+127 от текущего значения IP (1 байт).
Ближний (near) - в пределах того же сегмента (2 байта).
Дальний (far) - на произвольный адрес (4 байта).

ответ Метка занимает 1 байт, переход допустим в диапазоне от -128 до 127 байт от текущего значения IP

4. Какая команда выполняет переход, соответствующий условию "больше"?
   ответ JG

Команда	Описание	Состояние флагов для выполнения перехода
JO	Есть переполнение	OF = 1
JNO	Нет переполнение	OF = 0
JS	Есть знак	SF = 1
JNS	Нет знака	SF = 0
JE, JZ	Если равно/если ноль	ZF = 1
JNE, JNZ	Не равно/не ноль	ZF = 0
JP/JPE	Есть чётность / чётное	PF = 1
JNP/JPO	Нет чётности / нечётное	PF = 0
JCXZ	CX = 0	-

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JB JNAE JC	Если ниже Если не выше и не равно Если перенос	CF = 1	нет
JNB JAE JNC	Если не ниже Если выше или равно Если нет переноса	CF = 0	нет
JBE JNA	Если ниже или равно Если не выше	CF = 1 или ZF = 1	нет
JA JNBE	Если выше Если не ниже и не равно	CF = 0 и ZF = 0	нет

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JL JNGE	Если меньше Если не больше и не равно	SF <> OF	да
JGE JNL	Если больше или равно Если не меньше	SF = OF	да
JLE JNG	Если меньше или равно Если не больше	ZF = 1 или SF <> OF	да
JG JNLE	Если больше Если не меньше и не равно	ZF = 0 и SF = OF	да

5. Имеется описание следующего сегмента:

   DS SEGMENT

   ORG 100h

   I DW 0

   A DB 1

   DS ENDS

   Чему равно OFFSET A?
   Команда ORG сдвигает на заданное кол-во байт память, и все после нее будет записано с таким смещением.
   ответ 102h

6. В каких командах языка ассемблера применяется сегментный префикс?
   ответ При работе со значениями переменных

Сегментный префикс.

Pandas edited this page on 14 Jun 2017 · 1 revision

Конструкция вида

```
AS:X
```

где AS — какойто сегментный регистр (DS, ES, SS, CS), а X какая-то метка (ближний адрес) или другой регистр (SI, DI, SP, IP). Явно указывает, какой регистр мы используем для получения номера сегмента, к которому мы обращаемся.

Н-р: `MOV AX, ES:X`

7. Какая операция с сегментным регистром недопустима?
   ответ Загрузка константы

8. mov ax, [bx][si]+10
   ответ bx+si+10

9. Что такое неупакованное двоично-десятичное число?
   ответ Десятичная цифра, хранящаяся в байте

   Неупакованный двоично-десятичный тип — байтовое представление десятичной цифры от 0 до 9. **Неупакованные десятичные числа** хранятся как байтовые значения без знака по одной цифре в каждом байте.

10. Что такое упакованное двоично-десятичное число?
    ответ Две десятичные цифры, хранящиеся в полубайтах одного байта

    Десятичные числа — специальный вид представления числовой информации, в основу которого положен принцип кодирования каждой десятичной цифры числа группой из четырех бит. При этом каждый байт числа содержит одну или две десятичные цифры в так называемом двоично-десятичном коде (BCD — Binary-Coded Decimal). Микропроцессор хранит BCD-числа в двух форматах (рис. 3):

    - упакованном формате — в этом формате каждый байт содержит две десятичные цифры. Десятичная цифра представляет собой двоичное значение в диапазоне от 0 до 9 размером 4 бита. При этом код старшей цифры числа занимает старшие 4 бита. Следовательно, диапазон представления десятичного упакованного числа в одном байте составляет от 00 до 99;
    - неупакованном формате — в этом формате каждый байт содержит одну десятичную цифру в четырех младших битах. Старшие четыре бита имеют нулевое значение. Это так называемая зона. Следовательно, диапазон представления десятичного неупакованного числа в одном байте составляет от 0 до 9.

11. Какая команда осуществляет циклический сдвиг битов вправо

Логический, арифметический, циклический сдвиг. SAR, SAL, SHR, SHL, ROR, ROL, RCR, RCL

- SAL тождественна SHL
- SHR зануляет старший бит, SAR - сохраняет (знак)
- ROR, ROL - циклический сдвиг вправо/влево
- RCR, RCL - циклический сдвиг через CF

SAR - арифметический сдвиг вправо
SAL - арифметический сдвиг влево
SHR - логический сдвиг вправо
SHL - логический сдвиг влево
ROR - циклический сдвиг вправо
ROL - циклический сдвиг влево
RCR - циклический сдвиг вправо через перенос
RCL - циклический сдвиг влево через перенос

12. Что делает команда TEST?

Команда TEST

TEST <приёмник>, <источник>

- Аналог AND, но результат не сохраняется
- Выставляются флаги SF, ZF, PF

ответ Побитовое И без сохранения результата

13. Укажите,на какую величину уменьшится указатель вершины стека после выполнения следующих команд:

```
PUSH AX    ; - 2б
PUSH BX    ; - 2б
POP CX     ; + 2б
CALL F ; подпрограмма объявлена в том же сегменте -2б
```
=> уменьшится на 4

Стек

- LIFO/FILO (last in, first out) - последним пришёл, первым ушёл
- Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний
- SP - указатель на вершину стека
- В x86 стек "растёт вниз", в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента

Команды непосредственной работы со стеком

- PUSH <источник> - поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- POP <приёмник> - считать данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- PUSHA - поместить в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- POPA - загрузить регистры из стека (SP игнорируется)

14. Укажите, на какую величину изменится указатель вершины стека после выполнения следующей команды:

    RETN 4

    ## CALL - вызов процедуры,
    ## RET - возврат из процедуры

    CALL <операнд>

    - Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
    - Передаёт управление на значение аргумента.

    RET/RETN/RETF <число>

    - Загружает из стека адрес возврата, увеличивает SP
    - Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

    ответ на 6, так как 2 байта - адрес возврата, и еще 4 - операнд команды.

    RENF 4, то ответ будет 8, так как 4 байта возврат, так как длинный переход

15. укажите, какая характеристика соответствует команде NEG

NEG - изменение знака

NEG <приёмник>

Переводит число в дополнительный код.