1. Архитектура фон Неймана, принципы фон Неймана.

a. Процессор состоит из блоков **УУ** и **АЛУ**
b. УУ - дискретный конечный автомат. Структурно состоит из дешифратора команд (операций), регистра команд, узла вычислений текущего исполнительного адреса, счётчика команд (**регистр IP**). Указывает на смещение (адрес) инструкций в сегменте кода (1234:0100h сегмент/смещение).
    i. IP — 16-битный (младшая часть EIP)
    ii. EIP — 32-битный аналог (младшая часть RIP)
    iii. RIP — 64-битный аналог

c. АЛУ - под управлением УУ производит преобразование над данными (операндами). Разрядность операнда - длина машинного слова. (Машинное слово - машинно-зависимая величина, измеряемая в битах, равная разрядности регистров/шины данных)

Принципы фон Неймана:

d. Использование двоичной системы счисления в вычислительных машинах
e. Программное управление ЭВМ - все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов — команд. Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются в естественной последовательности, то есть в порядке их положения в программе.
f. Память компьютера используется не только для хранения данных, но и программ - команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования; то есть одно и то же значение в ячейке памяти может использоваться и как данные, и как команда, и как адрес в зависимости лишь от способа обращения к нему. Это позволяет производить над командами те же операции, что и над числами
g. Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы - причём процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые словами, и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек — адреса.
h. Возможность условного перехода в процессе выполнения программы.

Вопрос: насколько современный комп соответствует архитектуре фон Неймана?

Ответ: почти да, но в современной оперативной памяти есть некоторые поля только для данных, и только для команд

2. Машинные команды, машинный код. Понятие языка ассемблера.

Машинная команда - инструкция (в двоичном коде) из аппаратно определённого набора, которую способен выполнить процессор. Несмотря на отдельные отличия, системы машинных команд имеют много общего. Любая ЭВМ обязательно содержит следующие группы команд обработки информации.

1. **Команды передачи данных** (перепись), копирующие информацию из одного места в другое.

2. **Арифметические операции**, которым фактически обязана своим рождением вычислительная техника. Конечно, доля вычислительных действий в современном компьютере заметно уменьшилась, но они по-прежнему играют в программах важную роль.

3. **Логические операции**, позволяющие компьютеру производить анализ получаемой информации. После выполнения такой команды, с помощью условного перехода ЭВМ способна выбрать дальнейший ход выполнения программы. Простейшими примерами команд рассматриваемой группы могут служить сравнение, а также известные логические операции И, ИЛИ, НЕ (инверсия). Кроме того, к ним часто добавляется анализ отдельных битов кода, их сброс и установка.

4. **Сдвиги** двоичного кода. Для доказательства важности этой группы команд достаточно вспомнить правило умножения столбиком: каждое последующее произведение записывается в такой схеме со сдвигом на одну цифру влево.

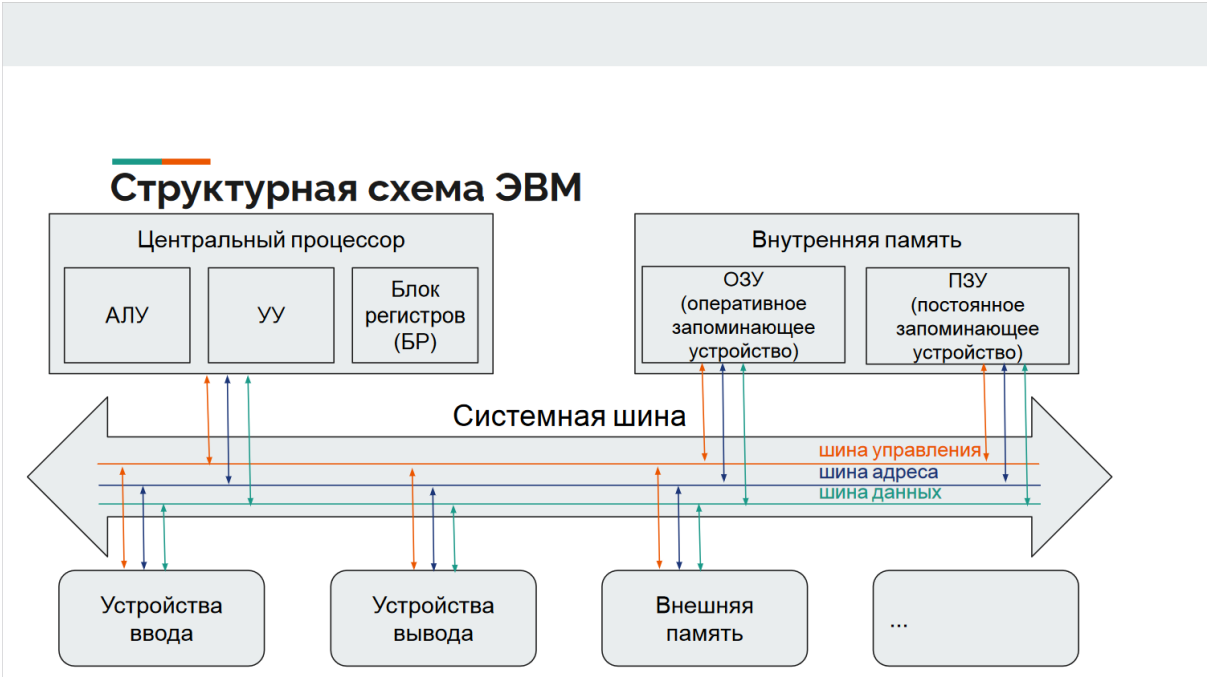
5. **Команды ввода и вывода** информации для обмена с внешними устройствами.

6. **Команды управления**, к которым прежде всего следует отнести условный и безусловный переход, а также команды обращения к подпрограмме (переход с возвратом).

Машинный код - система команд конкретной вычислительной машины, которая интерпретируется непосредственно процессором.

Язык ассемблера - машинно-зависимый язык программирования низкого уровня, команды которого прямо соответствуют машинным командам.

3. Виды памяти ЭВМ. Запуск и исполнение программы.



Байт - минимальная адресуемая единица памяти (8 бит).

Машинное слово - машинно-зависимая величина, измеряемая в битах, равная разрядности регистров/шины данных.

Параграф - 16 байт

Память делится на внешнюю и внутреннюю

К внутренней памяти относится:

- ОЗУ (оперативное запоминающее устройство)
- ПЗУ (постоянное запоминающее устройство). В ПЗУ хранится информация, которая записывается туда при изготовлении ЭВМ. Важнейшая микросхема ПЗУ - BIOS.

Выполнение программы:

1. Определение формата файла (.COM или .EXE, в случае 8086)
2. Чтение и разбор заголовка

3. Считывание разделов исполняемого модуля (файла) в ОЗУ по необходимым адресам.
4. Подготовка к запуску, если требуется. (установка регистров; настройка окружения, загрузка библиотек (см. 1 ЛР, 2 часть))
5. Передача управления на точку входа.
6. Дальше выполняются инструкции заданные в самой программе.

4. Сегментная модель памяти в архитектуре 8086.

Архитектура 8086 имеет четыре сегментных регистра: CS (code segment), SS (stack segment), DS (data segment), ES (extra segment). (Каждый сегментный регистр определяет адрес начала сегмента в памяти, при этом сегменты могут совпадать или пересекаться. По умолчанию регистр CS используется при выборке инструкций, регистр SS при выполнении операций со стеком, регистры DS и ES при обращении к данным)

Логический адрес записывают как сегмент:смещение (и те, и те в 16 с/с). В реальном режиме для вычисления физического адреса, адрес из сегмента сдвигают влево на 4 разряда (можно сказать, что просто приписывают 0 в конце или умножают на 16) и добавляют смещение. Например, логический адрес 7522:F139 дает физический адрес 84359.

На шину передается именно физический адрес. Если результат больше, чем 2^20 - 1, то 21 бит отбрасывают.

Такой режим работы процессора называют реальным режимом адресации процессора

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.

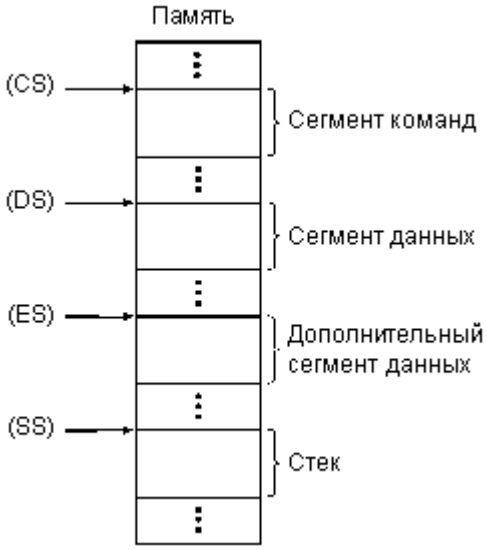
5. Процессор 8086. Регистры общего назначения.

Регистры общего назначения — это регистры данных, каждый из которых помимо хранения операндов и результатов операций имеет еще и свое специфическое назначение:

- регистр AX (*accumulator*) — умножение, деление, обмен с устройствами ввода/вывода (команды ввода и вывода);
- регистр BX (*base*) — базовый регистр в вычислениях адреса, часто указывает на начальный адрес (называемый базой) структуры в памяти;
- регистр CX (*count*) — счетчик циклов, определяет количество повторов некоторой операции;
- регистр DX (*data*) — определение адреса ввода/вывода, также может содержать данные, передаваемые для обработки в подпрограммы.

Эти определения, ни коим образом, не являются полными, и большую часть времени вы сами решаете, как использовать эти регистры общего назначения. Например, несмотря на то, что cx назван регистром-счетчиком, ничто не мешает использовать для подсчета регистр bx. *Однако в ряде случаев некоторые команды процессора 8086 требуют строго определенных регистров*.

6. Процессор 8086. Сегментные регистры. Адресация в реальном режиме. Понятие сегментной части адреса и смещения.



Сегментные регистры (CS, DS, SS, ES) определяют в памяти начала четырех 64 килобайтовых сегментов, которые называются *текущими сегментами*.

Программа может распределять более четырёх сегментов, но при этом для адресации дополнительных сегментов она должна перемещать соответствующие им правильные значения адресов между одним или несколькими сегментными регистрами.

Сегментные регистры строго специализированы. С помощью них нет возможности выполнять математические вычисления или хранить в них результаты других операций. Действительный порядок сегментов не обязательно совпадает с порядком, показанным на рисунке. Сегменты могут в любом порядке храниться в произвольных местах памяти.

- регистр CS (*Code Segment*) – содержит начальный адрес сегмента кода (начало машинного кода программы). Этот адрес плюс значение смещения в командном указателе (IP) определяет адрес команды, которая должна быть выбрана для выполнения;
- регистр DS (*Data Segment*) – содержит начальный адрес сегмента данных (переменных, строк и т.п. данных, которыми оперирует программа);
- регистр SS (*Stack Segment*) – содержит начальный адрес сегмента стека;
- регистр ES (*Extra Segment*) – является вспомогательным регистром, используется при некоторых операциях над строками. В большинстве программ в ES и DS содержатся одинаковые адреса, но он может упрощать некоторые операции связанные с этими регистрами.

Реальный режим работы - режим совместимости современных процессоров с 8086. Доступен 1 Мб памяти, то есть разрядность шины адреса - 20 разрядов. Физический адрес получается сложением адреса начала сегмента (на основе сегментного регистра) и смещения. Сегментный регистр хранит в себе старшие 16 разрядов (из 20) адрес начала сегмента. 4 младших разряда в адрес начала сегмента всегда нулевые. Говорят, что сегментный регистр содержит в себе номер параграфа начала сегмента.

7. Процессор 8086. Регистр флагов.

Флаги выставляются при выполнение операций, в основном арифметических. С помощью этих флагов можно определить что-нибудь, например было ли переполнение при последней выполненной операции.

Каждый флаг представляет собою 1 бит, выставляемый в 0 (флаг сброшен) или в 1 (флаг установлен). Не существует специальных команд, позволяющих обращаться к этому регистру напрямую.

Хотя разрядность регистра FLAGS 16 бит, реально используют не все 16. Остальные были зарезервированы при разработке процессора, но так и не были использованы.

Вот за что отвечает каждый бит в регистре 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	IO PL	IO PL	NT	-

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 чётности - 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 заём 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.

8. Команды пересылки данных

```
MOV <приемник>, <источник>.
```

- Приемник: РОН (регистр общего назначения), сегментный регистр, переменная (то есть ячейка памяти)
- Источник: непосредственный операнд (например, число), РОН, сегментный регистр, переменная

Нельзя загрузить в сегментный регистр значение непосредственно из памяти. Поэтому для этого используют промежуточный регистр (в начале лабы всегда так делали). Переменные не могут быть одновременно и источником, и приемником.

```
XCHG <операнд1>, <операнд2>
```

Обмен операндов между собой. Выполнятеся либо над двумя регистрами, либо регистр + переменная, но не две переменные сразу.

9. Команда сравнения.

```
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.

```
TEST <приемник>, <источник>
```

Аналог AND, но результат не сохраняется. Выставляются флаги SF, ZF, PF. Можно использовать для проверки на ноль, например TEST bx, bx

10. Команды условной и безусловной передачи управления

https://github.com/mRrvz/bmstu-asm/wiki/%5B%D0%AD%D0%9A%D0%97%D0%90%D0%9C%D0%95%D0%9D-17%5D-%D0%92%D0%B8%D0%B4%D1%8B-%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4%D0%BE%D0%B2.-%D0%A3%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5,-%D0%B1%D0%B5%D0%B7%D1%83%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5-%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4%D1%8B.-%D0%9A%D0%BE%D1%80%D0%BE%D1%82%D0%BA%D0%B8%D0%B9,-%D0%B1%D0%BB%D0%B8%D0%B6%D0%BD%D0%B8%D0%B9,-%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9-%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4.

вопрос №17

11. Арифметические команды

https://github.com/mRrvz/bmstu-asm/wiki/%5B%D0%AD%D0%9A%D0%97%D0%90%D0%9C%D0%95%D0%9D-09%5D-%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D1%8B-%D1%86%D0%B5%D0%BB%D0%BE%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D0%B9-%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D0%BA%D0%B8.

вопрос №9

12. Двоично-десятичная арифметика

https://github.com/mRrvz/bmstu-asm/wiki/%5B%D0%AD%D0%9A%D0%97%D0%90%D0%9C%D0%95%D0%9D-23%5D-%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D1%8B-%D0%B4%D0%B5%D1%81%D1%8F%

D1%82%D0%B8%D1%87%D0%BD%D0%BE%D0%B9-%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D0%BA%D0%B8

вопрос №23

13. Команды побитовых операций. Логические команды.

- `AND <приёмник>, <источник>` - побитовое "И"
- `OR <приёмник>, <источник>` - побитовое "ИЛИ"
- `XOR <приёмник>, <источник>` - побитовое исключающее "ИЛИ"
- `NOT <приёмник>` - инверсия
- `SHL <приёмник>, <счётчик>` - сдвиг влево (SAL - эквивалентная команда)
- `SHR <приёмник>, <счётчик>` - сдвиг вправо
- `SAR <приёмник>, <счётчик>` - сдвиг вправо с сохранением знакового бита
- `ROR <приёмник>, <счётчик>` - циклический сдвиг вправо
- `RCR <приёмник>, <счётчик>` - циклический сдвиг вправо, через флаг переноса
- `ROL <приёмник>, <счётчик>` - циклический сдвиг влево
- `RCL <приёмник>, <счётчик>` - циклический сдвиг влево, через флаг переноса

Все эти команды меняют регистр FLAGS.

14. Команды работы со строками.

https://github.com/mRrvz/bmstu-asm/wiki/%5B%D0%AD%D0%9A%D0%97%D0%90%D0%9C%D0%95%D0%9D-26%5D-%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D1%8B%D0%B5-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8.-%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D1%8B-%D0%BF%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%B5%D0%BD%D0%B8%D1%8F - вопрос №26

15. Команда трансляции по таблице.

```
XLAT <адрес (сегментный регистр)>
```

Помещает в AL байт из таблицы в памяти по адресу ES:BX (или ES:EBX) со смещением относительно начала таблицы равным AL. В качестве аргумента для XLAT в ассемблере можно указать имя таблицы, но эта информация никак не используется процессором и служит только в качестве комментария. Если он не нужен, можно применить форму записи XLATB.

если в сегменте данных, на который указывает регистр ES, было записано htable db "0123456789ABCDEF", то теперь AL содержит не число 0Ch, а ASCII-код буквы C. Разумеется, это преобразование разрешается выполнить посредством более компактного кода всего из трех арифметических команд, который будет рассмотрен в описании команды DAS, но с XLAT можно осуществить любые преобразования такого рода.

XLATB

Команда **xlatb** эквивалентна команде **xlat** МП 86 за исключением того, что для 32-разрядных приложений относительный адрес таблицы размещается в расширенном регистре EBX.

Короче говоря, XLATB -> AL = DS:[(E)BX + AL]

16. Команда вычисления эффективного адреса.

LEA <приёмник>, <источник>

Вычисляет эффективный адрес источника и помещает его в приёмник. Позволяет вычислить адрес, описанный сложным методом адресации (да и просто его загрузить).

Эффективный (текущий) адрес - это

```
БАЗА + СМЕЩЕНИЕ + ИНДЕКС
```

где БАЗА - это базовый адрес, находящийся в регистре (при 16-разрядной адресации могут использоваться только регистры BX или BP); СМЕЩЕНИЕ (или ОТКЛОНЕНИЕ - displacement) - это константа (число со знаком), заданная в команде; ИНДЕКС - значение индексного регистра (при 16-разрядной адресации могут использоваться только регистры SI или DI). С помощью команды LEA можно вычислить адрес переменной, которая описана сложным способом адресации (например, по базе с индексированием, что часто используется при работе с массивами и строками). Если адрес 32-разрядный, а ПРИЁМНИК - 16-разрядный, то старшая половина вычисленного адреса теряется. Если наоборот, ПРИЁМНИК - 32-разрядный, а адрес 16-разрядный, то вычисленное смещение дополняется нулями.

Иногда используется для быстрых арифметических вычислений:

```
lea bx, [bx+bx*4]

lea bx, [ax+12]
```

Эти вычисления занимают меньше памяти, чем соответствующие MOV и ADD, и не изменяют флаги.

17. Структура программы на языке ассемблера. Модули. Сегменты.

https://github.com/mRrvz/bmstu-asm/wiki/%5B%D0%AD%D0%9A%D0%97%D0%90%D0%9C%D0%95%D0%9D-12%5D-%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0-%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B-%D0%BD%D0%B0-%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5-%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0.-%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B8.-%D0%A1%D0%B5%D0%B3%D0%BC%D0%B5%D0%BD%D1%82%D1%8B - вопрос №12

про выравнивание: http://konishchevdmitry.blogspot.com/2010/01/blog-post.html

18. Подпрограммы. Объявление, вызов.

+

19. Подпрограммы. Возврат управления.

https://prog-cpp.ru/asm-proc/

Поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий.

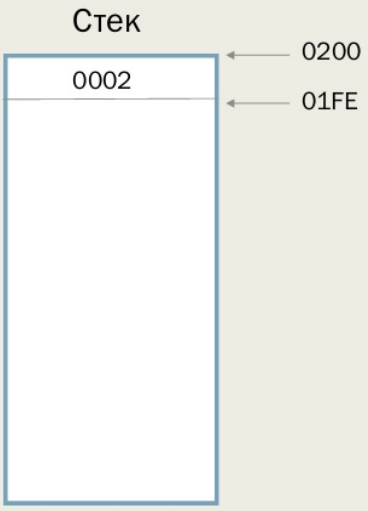
`CALL <операнд>` — передает управление на адрес <операнд>.

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

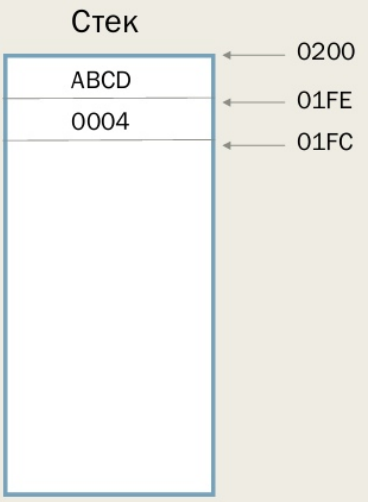
`RET/RETN/RETF <число>` — загружает из стека адерс возврата, увеличивая SP.

Если указать операнд, то можно очистить стек для очистки стека от параметров (<число> будет прибавлено к SP).

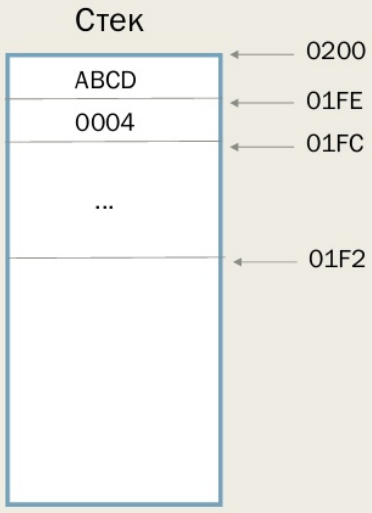
```
                    0. SP = 0200
0000: CALL P1       1. SP = 01FE
0002: MOV BX, AX
…
P1:
0123: MOV AX, 5
0125: RET           2. SP = 0200
```




```
                                        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
```



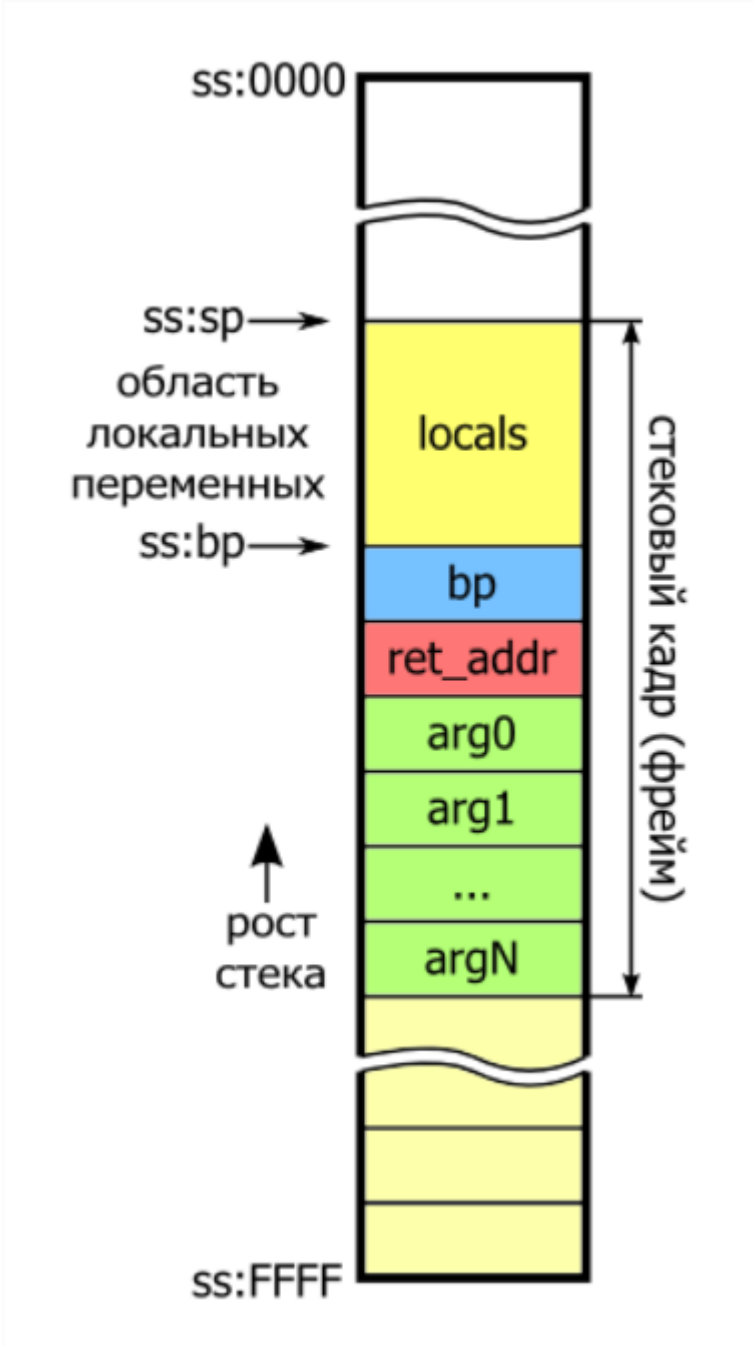

```
                                        0. SP = 0200
0000: PUSH ABCDh ;передача параметра    1. SP = 01FE
0002: CALL P1                           2. SP = 01FC
0004: MOV BX, AX
…
P1:
0123: MOV BP, SP ;ss:[bp] - адрес возврата
;ss:[bp+2] - параметр
0125: SUB SP, 10 ; ss:[bp-1 .. bp-10] - локальные переменные
                                        3. SP = 01F2
…
0221: ADD SP, 10                        4. SP = 01FC
0223: MOV AX, 5                         5. SP = 0200
0225: RET 2
```



20. Макроопределения.

Вопросы с 42 по 46

21. Стек. Аппаратная поддержка вызова подпрограмм.



https://github.com/mRrvz/bmstu-asm/wiki/%5B%D0%AD%D0%9A%D0%97%D0%90%D0%9C%D0%95%D0%9D-27%5D-%D0%A1%D1%82%D0%B5%D0%BA.-%D0%A0%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80%D1%8B,-%D1%81%D0%B2%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5-%D1%81%D0%BE-%D1%81%D1%82%D0%B5%D0%BA%D0%BE%D0%BC.-%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D1%8B-%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D0%B8-%D0%B8%D0%B7%D0%B2%D0%BB%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8%D0%B7-%D1%81%D1%82%D0%B5%D0%BA%D0%B0. №27

22. Прерывания. Обработка прерываний в реальном режиме работы процессора.

Прерывания

Прерывание - особая ситуация, когда выполнение текущей программы приостанавливается и управление передается 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.

Таблица векторов прерываний

Вектор прерывания - номер, который идентифицирует соответствующий обработчик прерываний. Векторы прерываний объединяются в таблицу векторов прерываний, содержащую адреса обработчиков прерываний. Располагается в самом начале памяти, начиная с адреса 0. Доступно 256 прерываний. Каждый вектор занимает 4 байта - полный адрес. Размер всей таблицы - 1 Кб.

Срабатывание прерывания

- Сохранение в текущий стек регистра флагов и адреса возврата (адреса следующей команды)
- Передача управления по адресу обработчика из таблицы векторов
- Настройка стека (возможно, обработчику прерываний нужен свой стек, потому что стек остается связан с той программой, которая работала до срабатывания прерывания; если обработчик сложный, то иногда такие обработчики перенастраивают стек)
- Повторная входимость (реентерабельность), необходимость запрета прерываний (Кузнецов: "таймер тикает, срабатывают прерывания. В какой-то момент прерывание тика таймера не успевает отработать до след тика, вызывается еще раз тоже прерывание и нужно обеспечить корректную работу в такой ситуации"; запрет прерывания можно делать только на короткий срок, иначе можно потерять данные (переполнение буфера клавиатуры, например))

Обработчик прерывания в реальном режиме

Располагается в самом начале памяти, начиная с адреса 0. Доступно 256 прерываний. Каждый вектор занимает 4 байта - полный адрес. Размер всей таблицы - 1 Кб.

Возврат из обработчика прерываний

IRET - используется для выхода из обработчика прерывания. Восстанавливает FLAGS, CS:IP. При необходимости выставить значение флага обработчик меняет его значение непосредственно в стеке.

23. Процессор 80386. Режимы работы. Регистры.

Процессор архитектуры x86 может работать в одном из пяти режимов и переключаться между ними очень быстро:

1. Реальный (незащищенный) режим (real address mode) — режим, в котором работал процессор 8086. В современных процессорах этот режим поддерживается в основном для совместимости с древним программным обеспечением (DOS-программами).

2. Защищенный режим (protected mode) — режим, который впервые был реализован в 80286 процессоре. Все современные операционные системы (Windows, Linux и пр.) работают в защищенном режиме. Программы реального режима не могут функционировать в защищенном режиме.

3. Режим виртуального процессора 8086 (virtual-8086 mode, V86) — в этот режим можно перейти только из защищенного режима. Служит для обеспечения функционирования программ реального режима, причем дает возможность одновременной работы нескольких таких программ, что в реальном режиме невозможно. Режим V86 предоставляет аппаратные средства для формирования виртуальной машины, эмулирующей процессор 8086. Виртуальная машина формируется программными средствами операционной системы. В Windows такая виртуальная машина называется VDM (Virtual DOS Machine — виртуальная машина DOS). VDM перехватывает и обрабатывает системные вызовы от работающих DOS-приложений.

4. Нереальный режим (unreal mode, он же big real mode) — аналогичен реальному режиму, только позволяет получать доступ ко всей физической памяти, что невозможно в реальном режиме.

5. Режим системного управления System Management Mode (SMM) используется в служебных и отладочных целях.

При загрузке компьютера процессор всегда находится в реальном режиме, в этом режиме работали первые операционные системы, например MS-DOS, однако современные операционные системы, такие как Windows и Linux переводят процессор в защищенный режим. Вам, наверное, интересно, что защищает процессор в защищенном режиме? В защищенном режиме процессор защищает выполняемые программы в памяти от взаимного влияния (умышленно

или по ошибке) друг на друга, что легко может произойти в реальном режиме. Поэтому защищенный режим и назвали защищенным.

отсюда: https://sites.google.com/site/sistprogr/lekcii1/lek2

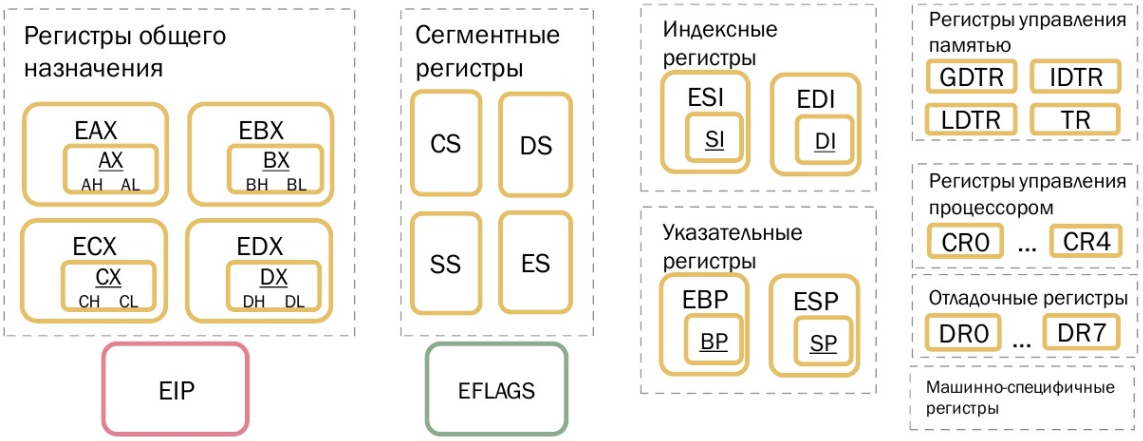
32-разрядные:

- регистры, кроме сегментных
- шина данных
- шина адреса (2^32 = 4ГБ ОЗУ)

Регистры

EDX = Extended DX (обращение к частям остается (DH, DL))

Добавлены регистры поддержки работы в защищенном режиме (обеспечение разделения доступа программ между собой, между программами и ОС и тд; эти регистры справа на картинке)



24. Математический сопроцессор. Типы данных.

Математический сопроцессор

Отдельное опциональное устройство на материнской плате, с 80486DX встроен в процессор.

Типы данных

- Целое слово (16 бит);

- Короткое целое (32 бита);
- Длинное слово (64 бита);
- Упакованное десятичное (80 бит);
- Короткое вещественное (32 бита);
- Длинное вещественное (64 бита);
- Расширенное вещественное (80 бит).

25. Математический сопроцессор. Регистры.

В сопроцессоре доступно 8 80-разрядных регистров (R0..R7).

- R0..R7, адресуются не по именам, а рассматриваются в качестве стека ST. ST соответствует регистру - текущей вершине стека, ST(1)..ST(7) - прочие регистры
- SR - регистр состояний, содержит слово состояния FPU. Сигнализирует о различных ошибках, переполнениях. Отдельные биты описывают и состояния регистров и в целом сигнализируют об ошибках (переполнениях и тп) при последней операции.
- CR - регистр управления. Контроль округления, точности (тоже 16 разрядный). Через него можно настраивать правила округления чисел и контроль точности (с помощью специальных битов устанавливать параметры, гибкие настройки)
- TW - 8 пар битов, описывающих состояния регистров: число (00), ноль (01), не-число (10), пусто (11) (изначально все пустые, проинициализированы единичками)
- FIP, FDP - адрес последней выполненной команды и её операнда для обработки исключений

26. Математический сопроцессор. Классификация команд.

https://github.com/mRrvz/bmstu-asm/wiki/%5B%D0%AD%D0%9A%D0%97%D0%90%D0%9C%D0%95%D0%9D-38%5D-%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9-%D1%81%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80.-%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0

%BA%D0%B0%D1%86%D0%B8%D1%8F-%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4 - вопрос №38

27. Расширения процессора. MMX. Регистры, поддерживаемые типы данных.

Расширение, которое было встроено для увеличения эффективности обработки больших потоков данных (простые операции над массивами однотипных данных (звук, изображения, видеопоток))

Типы данных MMX:

- учетверённое слово (64 бита);
- упакованные двойные слова (2);
- упакованные слова (4);
- упакованные байты (8).

Команды MMX перемещают упакованные данные в память или обычные регистры целиком, но арифметические и логические операции выполняются поэлементно.

8 64-битных регистров MM0..MM7 - мантиссы регистров FPU. При записи в MMn экспонента и знаковый бит заполняются единицами (2 байта, знак и экспонента). Пользоваться одновременно и FPU, и MMX не получится, требуется FSAVE+FRSTOR.

Насыщение - замена переполнения/антипереполнения превращением в максимальное/минимальное значение (Светлый цвет + светлый цвет максимум равно белый (но не будет переполнения и темного цвета))

28. Расширения процессора. MMX. Классификация команд.

1. Команды пересылки данных:

- пересылка двойных/учетверенных слов;
- упаковка со знаковым насыщением слов (приемник -> младшая половина приемника, источник -> старшая

половина приемника, в случае наличия значащих разрядов в отбрасываемых частях происходит насыщение);
- упаковка слов с беззнаковым насыщением, распаковка и объединение старших элементов источника и приемника через 1

2. Арифметические операции:

- поэлементное сложение (перенос игнорируется, побайтовое сложение)
- сложение/вычитание с насыщением
- беззнаковое сложение с насыщением
- вычитание (заем игнорируется)
- вычитание с насыщением
- старшее/младшее умножение (сохраняет старшую или младшую части результата в приемник)
- умножение и сложение (перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших)

3. Команды сравнения:

- проверка на равенство (Если пара равна - соответствующий элемент приёмника заполняется единицами, иначе - нулями)
- проверка на больше (Если элемент приёмника больше, то заполняется единицами, иначе - нулями)

4. Логические операции:

- логическое И
- логическое ИЛИ
- логическое НЕ-И (штрих шеффера)
- XOR

5. Сдвиговые операции:

- логический влево
- логический вправо
- арифметический вправо

29. Расширения процессора. SSE. Регистры, поддерживаемые типы данных

Регистры:

- 8 128-разрядных регистров
- свой регистр флагов

Основной тип - вещественные одинарной точности (32 бита, в 1 регистре 4 числа)

Целочисленные команды работают с регистрами MMX

Команд больше чем в MMX, типы:

- Пересылки
- Арифметические
- Сравнения
- Преобразования типов
- Логические
- Целочисленные
- Упаковки
- Управления состоянием
- Управления кэшированием

30. Расширения процессора. SSE. Классификация команд

Классификация команд:

1. Команды пересылки данных:

- пересылка двойных/учетверенных слов;
- упаковка со знаковым насыщением слов (приемник -> младшая половина приемника, источник -> старшая половина приемника, в случае наличия значащих разрядов в отбрасываемых частях происходит насыщение);

- упаковка слов с беззнаковым насыщением, распаковка и объединение старших элементов источника и приемника через 1

2. Арифметические операции:

- поэлементное сложение (перенос игнорируется, побайтовое сложение)
- сложение/вычитание с насыщением
- беззнаковое сложение с насыщением
- вычитание (заем игнорируется)
- вычитание с насыщением
- старшее/младшее умножение (сохраняет старшую или младшую части результата в приемник)
- умножение и сложение (перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших)

3. Команды сравнения:

- проверка на равенство (Если пара равна - соответствующий элемент приёмника заполняется единицами, иначе - нулями)
- проверка на больше (Если элемент приёмника больше, то заполняется единицами, иначе - нулями)

4. Логические операции:

- логическое И
- логическое ИЛИ
- логическое НЕ-И (штрих шеффера)
- XOR

5. Сдвиговые операции:

- логический влево
- логический вправо
- арифметический вправо