1. Архитектура фон Неймана, принципы фон Неймана

Архитектура фон Неймана — широко известный принцип совместного хранения команд и данных в памяти компьютера. Вычислительные машины такого рода часто обозначают термином «машина фон Неймана», однако соответствие этих понятий не всегда однозначно. В общем случае, когда говорят об архитектуре фон Неймана, подразумевают принцип хранения данных и инструкций в одной памяти.

- Процессор состоит из блоков УУ и АЛУ
- УУ - дискретный конечный автомат.

Структурно состоит из: дешифратора команд (операций), регистра команд, узла вычислений текущего исполнительного адреса, счетчика команд (регистр IP).

- АЛУ - под управлением УУ производит преобразование над данными (операндами).

Разрядность операнда - длина машинного слова. (Машинное слово - машинно-зависимая величина, измеряемая в битах, равна разрядности регистров/шины данных)

Принципы фон Неймана

1. Использование двоичной с/с в вычислительных машинах.
2. 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 порядке их положения в программе. При необходимости, с помощью специальных команд, эта последовательность может быть изменена.
3. Принцип однородности памяти. Память используется не только для хранения данных, но и для 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.
4. Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы.

- Структурно основная память состоит из пронумерованных ячеек, 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

5. Возможность условного перехода в процессе выполнения программы.

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

Машинный код

Машинный код или машинный язык представляет собой набор инструкций, выполняемых непосредственно центральным процессором компьютера (CPU). Каждая команда выполняет очень конкретную задачу, например, загрузки (load), перехода (jump) или элементарной арифметической или логической операции для единицы данных в регистре процессора или памяти. Каждая программа выполняется непосредственно процессором и состоит из ряда таких инструкций.

Машинный код можно рассматривать как самое низкоуровневое представление скомпилированной или собранной компьютерной программы или в качестве примитивного и аппаратно-зависимого языка программирования. Писать программы непосредственно в машинном коде возможно, однако это утомительно и подвержено ошибкам, так как необходимо управлять отдельными битами и вычислять числовые адреса и константы вручную. По этой причине машинный код практически не используется для написания программ.

Исходный код, написанный на языке высокого уровня транслируется в исполняемый машинный код с помощью интерпретатора, компилятора либо линкеры.

- Каждый процессор или семейство процессоров имеет свой собственный набор инструкций машинного кода. Говорят, что процессор A совместим с процессором B, если процессор A полностью «понимает» машинный код процессора B. Если процессоры A и B имеют некоторое подмножество инструкций, по которым они взаимно совместимы, то говорят, что они одной архитектуры. Таким образом, набор команд является специфическим для одного класса процессоров. Новые процессоры одной архитектуры часто включают в себя все инструкции предшественника и могут включать дополнительные.

  Системы также могут отличаться в других деталях, таких как расположение памяти, операционные системы или периферийные устройства. Поскольку программа обычно зависит от таких факторов, различные системы, как правило, не запустят один и тот же машинный код, даже если используется тот же тип процессора.

- Исполняемый файл — файл, содержащий программу в том виде, в котором она может быть исполнена компьютером.
  Стадии получения: компиляция + линковка (компоновка).
  Компилятор - программа для преобразования исходного текста другой программы на определенном ЯП в объектный модуль.
  Линковщик (компоновщик) - связывает несколько объектных файлов в исполняемый файл.

Понятие языка ассемблера.

Гораздо более читаемым представлением машинного языка называется язык ассемблера, использующий мнемонические коды для обозначения инструкций машинного кода, а не с помощью числовых значений.
Язык ассемблера - машинно-зависимый язык программирования низкого уровня, команды которого прямо соответствуют машинным командам.

- Компьютерная программа представляет собой последовательность команд, которые выполняются процессором. В то время как простые процессоры выполняют инструкции один за другим.
- Абсолютный и позиционно-независимый код
- **Абсолютный код** — программный код, пригодный для прямого выполнения процессором, то есть код, не требующий дополнительной обработки (например, разрешения ссылок между различными частями кода или привязки к адресам в памяти, обычно выполняемой загрузчиком программ). Примерами абсолютного кода являются исполняемые файлы в формате .COM и загрузчик ОС. Часто абсолютный код понимается в более узком смысле как позиционно-зависимый код (то есть код, привязанный к определенным адресам памяти).
- **Позиционно-независимый код** — программа, которая может быть размещена в любой области памяти, так как все ссылки на ячейки памяти в ней относительные (например, относительно счетчика команд). Такую программу можно переместить в другую область памяти в любой момент, в отличие от перемещаемой программы, которая хотя и может быть загружена в любую область памяти, но после загрузки должна оставаться на том же месте.

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).

3. Виды памяти ЭВМ. Запуск и исполнение программы.

Виды памяти

Байт - минимальная адресуемая единица памяти (8 бит).
word - 16 бит (2 байта)
dword - 32 бит (4 байта)
Машинное слово - машинно-зависимая величина, измеряемая в битах, равная разрядности регистров/шины данных.
Параграф - 16 байт

Память компьютера делится на внешнюю (основную) и внутреннюю.

К внутренней памяти относятся:

1. Оперативная память — это устройства, где размещены данные, который процессор обрабатывает в определенный промежуток времени. Из нее процессор берет программы и исходные данные для обработки, в нее он записывает полученные результаты. Название «оперативная» эта память получила потому, что она работает очень быстро, так что процессору практически не приходится ждать при чтении данных из памяти или записи в память. При этом выполняется следующее условие: в любой момент существует условие работы с любой ячейкой оперативной памяти. В оперативной памяти сохраняется временная информация, которая изменяется по мере выполнения процессором различных операций, таких как запись, считывание, сохранение. При отключении компьютера вся информация, которая находилась в оперативной памяти исчезает, если она не была сохранена на других носителях информации.

2. Регистры — это сверхскоростная память процессора. Они сохраняют адрес команды, саму команду, данные для ее выполнения и результат.

3. *Кэш-память.* Для ускорения доступа к оперативной памяти на быстродействующих компьютерах используется специальная кэш-память, которая располагается как бы «между микропроцессором и оперативной памятью и хранит копии наиболее часто используемых участков оперативной памяти.

При обращении микропроцессора к памяти сначала производится поиск нужных данных в кэш-памяти. Поскольку время доступа к кэш-памяти в несколько раз меньше, чем к обычной памяти, а в большинстве случаев необходимые микропроцессору данные уже содержатся в кэш-памяти, среднее время доступа к памяти уменьшается.

4. BIOS постоянная память, в которую данные занесены при изготовлении. Как правило, эти данные не могут быть изменены. Выполняемые на компьютере программы могут только их считывать.

5. Кроме оперативной памяти существует ещё и постоянная память (ПЗУ). Её главное отличие от ОЗУ — невозможность в процессе работы изменить состояние ячеек ПЗУ. В компьютере в постоянной памяти хранятся программы для проверки оборудования компьютера, инициирования загрузки ОС и выполнения базовых функций по обслуживанию устройств компьютера. Внешняя память рассчитана на длительное хранение программ и данных. Она реализуется с помощью специальных устройств, которые в зависимости от способов записи и считывания делятся на магнитные, оптические и магнитооптические.

Внешняя память

Основными характеристиками внешней памяти являются ее объем, скорость обмена информацией, способ и время доступа к данным. К внешней памяти принадлежат также

накопители на гибких дисках (дискетах). Наиболее распространенными являются дискеты диаметром 3,5 дюйма.

Функции памяти:

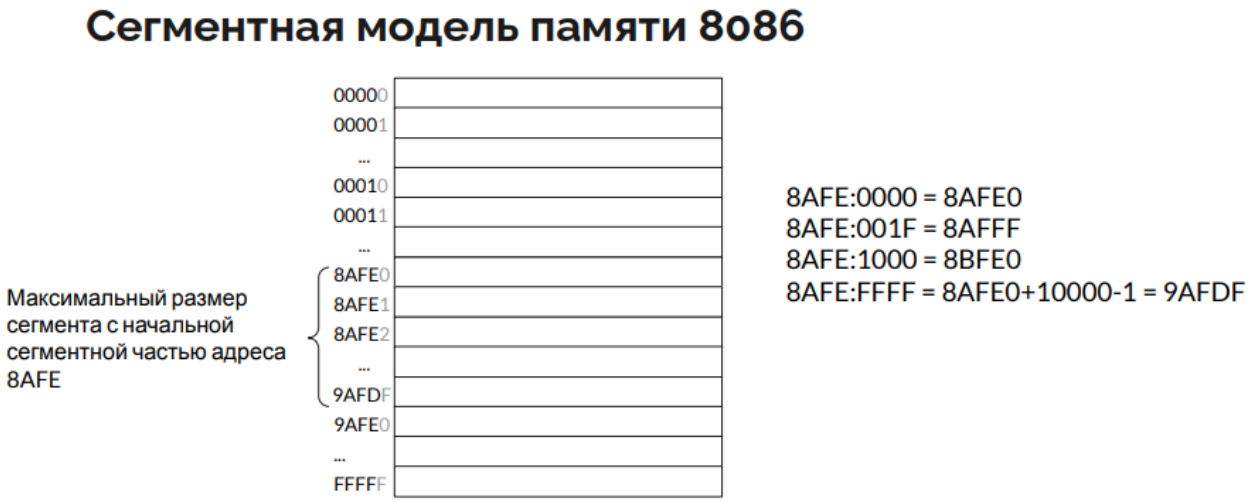
- приём информации из других устройств;
- запоминание информации;
- выдача информации по запросу в другие устройства машины.

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)
5. Передача управления на точку входа.

Дальше выполняются инструкции заданные в самой программе.

4. Сегментная модель памяти в архитектуре 8086.



Все байты памяти пронумерованы в один большой список.
Какой то сегмент начинается с адреса какого то. Сегментная часть адреса у него есть и первый байт памяти - смещение.

Сегментом называется условно выделенная область адресного пространства определённого размера, а смещением — адрес ячейки памяти относительно начала сегмента.

Сегментная адресация памяти — схема логической адресации памяти компьютера в архитектуре x86. Линейный адрес конкретной ячейки памяти, который в некоторых режимах работы процессора будет совпадать с физическим адресом, делится на две части: сегмент и смещение. Размер сегмента зависит от того сколько мы в него записываем. Максимальный размер сегмента - 2 в 16 степени (64кб). Может 64кб, но обычно меньше.

Базой сегмента называется линейный адрес, т.е. адрес относительно всего объёма памяти, который указывает на начало сегмента в адресном пространстве. Начало каждого

последующего сегмента смещено относительно базы предыдущего на минимальный размер сегмента, то есть на 16 байт (параграф). В результате получается сегментный (логический) адрес, который соответствует линейному адресу, база сегмента+смещение и выставляется процессором на шину адреса.

Сегменты могут частично перекрывать друг друга. (Например, байт 17 сегмента 2 — это также и байт (1 = 17 - 16) сегмента 3, и байт 33 = 17 + 16 сегмента 1.

- Селектор - число 16-битное, однозначно определяющее сегмент. (начало сегмента)
  Селектор загружается в сегментные регистры.
  В реальном и защищённом режимах x86-процессора функционирование сегментной адресации отличается.

- В реальном режиме процессора всё адресное пространство делится на одинаковые сегменты размером 65536 байт (2^16 байт)

  Селектор 16-разрядный и задаёт номер сегмента. Учитывая, что сегменты следуют друг за другом с постоянным интервалом в 2^4=16 байт, очень легко выяснить линейный адрес сегмента, умножая его на 16 (или сдвигая на 4 бита влево).

5. Процессор 8086. Регистры общего назначения.

Процессор 8086

Регистров всего в 8086 14.

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.
Регистры общего назначения: AX, BX, CX, DX.
Каждый из регистров имеет старшую и младшую часть, по 1 байту на каждый.
При использовании регистров общего назначения, можно обратится к каждым 8 битам (байту) по-отдельности, используя вместо *X - *H или *L (например, для AX: AH и AL).

AX (аккумулятор)
Аккумулятор - умножение, деление, обмен с устройствами ввода/вывода (команды ввода и вывода
Регистр часто используется для хранения результата действий, выполняемых над двумя операндами. Например, используется при MUL и DIV (умножение и деление)
Верхние 8 бит (1 байт) — AH
Нижние 8 бит (1 байт) — AL

BX (база)
Базовый регистр в вычислениях адреса, часто указывает на начальный адрес (называемый базой) структуры в памяти
Используется для адресации по базе.
Верхние 8 бит (1 байт) — BH
Нижние 8 бит (1 байт) — BL

CX (счётчик)
Используется как счетчик циклов, определяет количество повторов некоторой оперции
Верхние 8 бит (1 байт) — CH
Нижние 8 бит (1 байт) — CL

DX (регистр данных)
Определение адреса ввода/вывода, также может содержать данные, передаваемые для обработки в подпрограммы.
Если при выполнении действий над двумя операндами, результат не помещается в AX, регистр DX получает старшую часть результата.
Верхние 8 бит (1 байт) — DH
Нижние 8 бит (1 байт) — DL

- 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 обращаться к каждому из байтов по-отдельности

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

Сегментные регистры

- Сегмент кода (CS) — содержит все команды и инструкции, которые должны быть выполнены. 16-битный регистр.
- Сегменты данных (DS) — содержит данные, константы и рабочие области. 16-битный регистр.
- Сегмент стека (SS) — содержит данные и возвращаемые адреса процедур или подпрограмм. Он представлен в виде **структуры данных «Стек»**.

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.

Указатель команды: IP.
Этот регистр, содержит адрес-смещение следующей команды, относительно сегмента CS.
Связан с CS как CS:IP
Предположим, что в CS хранится адрес 2CB5H, а в регистре IP хранится смещение 123H.
Таким образом, адрес следующей команды 2CB50H + 123H = 2CC73H.

Адресация в реальном режиме

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.

Каждый регистр содержит адрес для разных сегментов программы, с помощью них и происходит доступ к этим сегментам.

Логический адрес записывают как сегмент:смещение (и те, и те в 16 с/с).
[SEG]:[OFFSET] => физический адрес:
1. В реальном режиме для вычисления физического адреса, SEG, то есть адрес из сегмента сдвигают влево на 4 разряда(или умножить на 16, что тождественно)
2. К результату прибавить OFFSET смещение

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.

!!!!!!!!!!!!Другие способы адресации

- Регистровая адресация (mov ax, bx)
- Непосредственная адресация (mov ax, 2)
- Прямая адресация (mov ax, ds:0032) (метка во время компиляции преобразуется в прямую)
- Косвенная адресация (mov ax, [bx]). В 8086 допустимы BX, BP, SI, DI
- Адресация по базе со сдвигом (mov ax, [bx]+2; mov ax, 2[bx]).
- Адресация по базе с индексированием (допустимы BX+SI, BX+DI, BP+SI, BP+DI):
    - mov ax, [bx+si+2]
    - mov ax, [bx][si]+2
    - mov ax, [bx+2][si]
    - mov ax, [bx][si+2]
    - mov ax, 2[bx][si]

!!!!!!!!!!!!!!!!!!!!!!

Понятие сегментной части адреса и смещения.

Сегментная адресация памяти — схема логической адресации памяти компьютера в архитектуре x86. Линейный адрес конкретной ячейки памяти, который в некоторых режимах работы процессора будет совпадать с физическим адресом, делится на две части: сегмент и смещение. Размер сегмента зависит от того сколько мы в него записываем. Максимальный размер сегмента - 2 в 16 степени (64кб). Может 64кб, но обычно меньше.

Базой сегмента называется линейный адрес, т.е. адрес относительно всего объёма памяти, который указывает на начало сегмента в адресном пространстве. Начало каждого

последующего сегмента смещено относительно базы предыдущего на минимальный размер сегмента, то есть на 16 байт (параграф). В результате получается сегментный (логический) адрес, который соответствует линейному адресу, база сегмента+смещение и выставляется процессором на шину адреса.

7. Процессор 8086. Регистр флагов.

Регистры флагов в 8086.

Флаги - выставляются при выполнение операций, в основном арифметических. С помощью этих флагов можно определить что-нибудь определить, 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	IOPL	IOPL	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 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 (overflow flag) - 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, нам не нужно пока.
■ NT (nested task) - флаг вложенности задач - а это на 286, нам не нужно пока.

8. Команды пересылки данных

Базовая команда пересылки данных. Копирует содержимое источника в приемник, источник не изменяется. Команда MOV действует аналогично операторам присваивания из языков высокого уровня.

В качестве источника для MOV могут использоваться: число (непосредственный операнд), регистр общего назначения, сегментный регистр или переменная (то есть операнд, находящийся в памяти); в качестве приемника: регистр общего назначения, сегментный регистр (кроме CS) или переменная. Оба операнда должны быть одного и того же размера - байт, слово или двойное слово. Мы не можем выполнять пересылку данных с помощью MOV из одной переменной в другую, из одного сегментного регистра в другой и нельзя помещать в сегментный регистр непосредственный операнд - эти операции выполняют двумя командами MOV (из сегментного регистра в обычный и уже из него в другой сегментный) или парой команд PUSH/POP.

Команда пересылки данных
MOV <приемник>, <источник>.

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

Переменные не могут быть одновременно и источником, и приемником.

XCHG <операнд 1>, <операнд 2>
Обмен операндов между собой. Выполняется либо над двумя регистрами, либо регистр + переменная.
xor ax, ax == 0 занулить регистр

9. Команда сравнения
Сравнивает приемник и источник и устанавливает флаги. Действие осуществляется путем вычитания источника из приемника, причем результат вычитания никуда не записывается. Единственным следствием работы этой команды оказывается изменение флагов CF, OF, SF, ZF, AF и PF. Обычно команду CMP используют вместе с командами условного перехода (Jcc), условной пересылки данных (CMOVcc) или условной установки байтов (SETcc), которые позволяют применить результат сравнения, не обращая внимания на детальное значение каждого флага.

CMP <приемник>, <источник>

Источник - число, регистр или переменная.

Приемник - регистр или переменная; не может быть переменной одновременно с источником.

Тип операндов	Мнемокод команды	Критерий перехода	Значения флагов для перехода
Любые	JE	Операнд_1=операнд_2	ZF=1
Любые	JNE	Операнд_1<>операнд_2	ZF=0
Со знаком	JL/JNGE	Операнд_1<операнд_2	SF<>OF
Со знаком	JLE/JNG	Операнд_1<=операнд_2	SF<>OF или ZF=1
Со знаком	JG/JNLE	Операнд_1>операнд_2	SF=OF и ZF=0
Со знаком	JGE/JNL	Операнд_1>=операнд_2	SF=OF
Без знака	JB/JNAE	Операнд_1<операнд_2	CF=1
Без знака	JBE/JNA	Операнд_1<=операнд_2	CF=1 или ZF=1
Без знака	JA/JNBE	Операнд_1>операнд_2	CF=0 и ZF=0
Без знака	JAE/JNB	Операнд_1=>операнд_2	CF=0

Мнемоника	*Описание*	*Состояние флагов*
JZ	Переход, если нуль	ZF = 1
JNZ	Переход, если не нуль	ZF = 0
JC	Переход, если перенос	CF = 1
JNC	Переход, если нет переноса	CF = 0
JO	Переход, если переполнение	OF = 1
JNO	Переход, если нет переполнения	OF = 0
JS	Переход, если флаг знака установлен	SF = 1
JNS	Переход, если флаг знака сброшен	SF = 0
JP	Переход, если флаг четности установлен	PF = 1
JNP	Переход, если флаг четности сброшен	PF = 0

TEST <приемник>, <источник>

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

test ax,ax сравнение с 0 равенство единице dec ax

CMPXCHG <приемник>, <источник> - Сравнить и обменять между собой.
Сравнивает значения, содержащиеся в AL, AX, EAX (в зависимости от размера операндов), с приемником (регистром). То есть, источник - всегда регистр, приемник может быть и регистром, и переменной.
Если они равны, информация из источника копируется в приемник и флаг ZF устанавливается в 1, в противном случае содержимое приемника копируется в AL, AX, EAX и флаг ZF устанавливается в 0. Остальные флаги определяются по результату операции сравнения, как после CMP.

10. 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.

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)

Для короткого и ближнего переходов непосредственный операнд (число) прибавляется к IP. Регистры и переменные заменяют старое значение в IP (CS:IP).

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.

TEST <приемник>, <источник>
Аналог AND, но результат не сохраняется. Выставляются флаги SF, ZF, PF.

Модификатор может принимать следующие значения:

- **near ptr** — прямой переход на метку внутри текущего сегмента кода. !!!!!!Модифицируется только регистр eip/ip (в зависимости от заданного типа сегмента кода use16 или use32) на основе указанного в команде адреса (метки) или выражения, использующего символ извлечения значения СчА — $;

- **far ptr** — прямой переход на метку в другом сегменте кода.
  !!!!!! Адрес перехода задается в виде непосредственного операнда или адреса (метки) и состоит из 16-битного селектора и 16/32-битного смещения, которые загружаются, соответственно, в регистры cs и ip/eip;
- **word ptr** — косвенный переход на метку внутри текущего сегмента кода.
  !!!!!!Модифицируется (значением смещения из памяти по указанному в команде адресу, или из регистра) только eip/ip. Размер смещения 16 или 32 бит;
- **dword ptr** — косвенный переход на метку в другом сегменте кода.
  !!!!!!!!!!Модифицируются (значением из памяти — и только из памяти, из регистра нельзя) оба регистра, cs и eip/ip. Первое слово/двойное слово этого адреса представляет смещение и загружается в ip/eip; второе/третье слово загружается в cs.

Команда безусловного перехода jmp

Синтаксис команды безусловного перехода

jmp [модификатор] адрес_перехода - безусловный переход без сохранения информации о точке возврата.
Адрес_перехода адрес в виде метки либо адрес области памяти, в которой находится указатель перехода.

Виды переходов для команды JMP
● short (короткий) -128 .. +127 байт
● near (ближний) в том же сегменте (без изменения регистра CS)
● far (дальний) в другой сегмент (с изменением значения в регистре CS)
● Для короткого и ближнего переходов непосредственный операнд (константа) прибавляется к IP
● Операнды - регистры и переменные заменяют старое значение в IP (CS:IP)
Их различия определяются дальностью перехода и способом задания целевого адреса.

Дальность перехода определяется местоположением операнда *адреса_перехода*. Он может находиться в текущем сегменте кода либо другом сегменте. В первом случае переход называется *внутрисегментным*, или *близким*, во втором — *межсегментным*, или *дальним*.
!!!!!!!!Внутрисегментный переход предполагает, что изменяется только содержимое регистра !!!!!!!!**eip/ip**.
Можно выделить три варианта внутрисегментного использования команды jmp:

- прямой короткий;
- прямой;
- косвенный.

!!!!Процедура, часто называемая также *подпрограммой*, — это основная функциональная единица декомпозиции (разделения на несколько частей) некоторой задачи.

Для описания последовательности команд в виде процедуры в языке ассемблера используются две директивы: **PROC** и **ENDP**.

!!!!!

11. 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.

- Команда выполняет арифметическое сложение приемника и источника, помещает сумму в приемник, не изменяя содержимое источника. Приемник может быть регистром или переменной, источник - числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника, и для приемника. Команда ADD никак не различает числа со знаком и без знака, но, употребляя значения флагов CF (перенос при сложении чисел без знака), OF (перенос при сложении чисел со знаком) и SF (знак результата), разрешается применять ее и для тех, и для других.

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.

$11111111_2 + 00000001_2 = (1)00000000_2$ (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

- Вычитает источник из приемника и помещает разность в приемник. Приемник может быть регистром или переменной, источник - числом, регистром или переменной, но нельзя использовать переменную одновременно и для источника и для приемника. Точно так же, как и команда ADD, ! между числами со знаком и без знака, но флаги позволяет тех, и для других.

```
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 <приемник>, <источник>. Умножает источник на приемник, результат в приемник.
3. IMUL <приёмник>, <источник1>, <источник2>. Умножает источник 1 на источник 2, результат в приемник.

Флаги: OF, CF

- Выполняет умножение содержимого источника (регистр или переменная) и регистра AL, AX, EAX (в зависимости от размера источника) и помещает результат в AX, DX:AX, EDX.-EAX соответственно. Если старшая половина результата (AH, DX, EDX) содержит только нули (результат целиком поместился в младшую половину), флаги CF и OF устанавливаются в 0, иначе - в 1. Значение остальных флагов (SF, ZF, AF и PF) не определено.
-

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.

- Выполняет целочисленное деление без знака AL, AX или EAX (в зависимости от размера источника) на источник (регистр или переменная) и помещает результат в AL, AX или EAX, а остаток - в AH, DX или EDX соответственно. Результат всегда округляется в сторону нуля, абсолютное значение остатка меньше абсолютного значения делителя. Флаги CF, OF, SF, ZF, AF и PF после этой команды не определены, а переполнение или деление на ноль вызывает исключение #DE (ошибка при делении) в защищенном режиме и прерывание 0-в реальном.

INC <приемник> — увеличивает приемник на 1.

- Увеличивает приемник (регистр или переменная) на 1. Единственное отличие этой команды от ADD приемник,1. состоит в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом сложения

DEC <приемник> — уменьшает приемник на 1.

- Уменьшает приемник (регистр или переменная) на 1. Единственное отличие этой команды от SUB приемник,! заключается в том, что флаг CF не затрагивается. Остальные арифметические флаги (OF, SF, ZF, AF, PF) устанавливаются в соответствии с результатом вычитания

Меняют флаги: OF, SF, ZF, AF, PF

NEG <приемник> — меняет знак приемника. Переводит число в дополнительный код.

- Выполняет над числом, содержащимся в приемнике (регистр или переменная), операцию дополнения до двух. Эта операция эквивалентна обращению знака операнда, если рассматривать его как число со знаком. Если приемник равен нулю, флаг CF устанавливается в 0, иначе - в 1. Остальные флаги (OF, SF, ZF, AF, PF) назначаются в соответствии с результатом операции

12. Двоично-десятичная арифметика.

Процессоры Intel поддерживают операции с двумя форматами десятичных чисел: неупакованное двоично-десятичное число - байт, принимающий значения от 00 до 09h, и

упакованное двоично-десятичное число - байт, принимающий значения от 00 до 99h. Все обычные арифметические операции над такими числами приводят к неправильным результатам. Например, если увеличить 19h на 1, то получится число lAh, а не 20h. Для коррекции результатов арифметических действий над двоично-десятичными числами используются приведенные ниже команды.

Упакованный BCD-формат - это упакованное двоично-десятичное число - байт от 00h до 99h (цифры A..F не задействуются).

DAA. Назначение: BCD-коррекция после сложения
Команда DAA (Decimal Adjust AL after Addition) позволяет получать результат сложения упакованных двоично-десятичных данных в таком же упакованном BCD-формате. То есть она корректирует после сложения, пример:

```
mov AL,71H   ; AL = 0x71h
add AL,44H   ; AL = 0x71h + 0x44h = 0xB5h
daa          ; AL = 0x15h
             ; CF = 1 - перенос является частью результата 71 + 44 = 115
```

DAA выполняет следующие действия:
1. Если младшие четыре бита AL больше 9 или флаг AF = 1, то AL увеличивается на 6, CF устанавливается, если при этом сложении произошел перенос, и AF устанавливается в 1.
2. Иначе AF = 0.
3. Если теперь старшие четыре бита AL больше 9 или флаг CF = 1, то AL увеличивается на 60h и CF устанавливается в 1.
4. Иначе CF = 0.

ASCII-формат - это неупакованное двоично-десятичное число (байт от 00h до 09h).

Команда AAS (ASCII Adjust After Subtraction) позволяет преобразовать результат вычитания двоично-десятичных данных в ASCII-формат. Для этого команда AAS должна выполняться после команды двоичного вычитания SUB, которая помещает однобайтный результат в регистр AL. Если был заем, будет вычитание 1 из AH.
Для того, чтобы преобразовать содержимое регистра AL к ASCII-формату, необходимо после команды AAS выполнить команду OR AL,0x30h (то есть сделать читаемым числом).
При положительном результате вычитания это выглядит следующим образом:

```
sub AH,AH    ; очистка AH
mov AL,'9'   ; AL = 0x39h
sub AL,'3'   ; AL = 0x39h - 0x33h = 0x06h
aas          ; AX = 0x0006h
or AL,30H    ; AL = 0x36h = '6'
```

при вычитании с получением результат меньше нуля:

```
sub AH,AH    ; очистка AH
mov AL,'3'   ; AL = 0x33h
sub AL,'9'   ; AL = 0x33h - 0x39h = 0xFAh
aas          ; AX = 0xFF04h
or AL,30H    ; AL = 0x34h = '4' (хз почему)
```

AAM imm8

Команда AAM (ASCII Adjust AX After Multiply) позволяет преобразовать результат умножения неупакованных двоично-десятичных данных в ASCII-формат. Для этого команда AAM должна выполняться после команды беззнакового умножения MUL (но не после команды умножения со знаком IMUL), которая помещает двухбайтный результат в регистр AX. Команда AAM распаковывает результат умножения, содержащийся в регистре AL, деля его на второй байт кода операции ib. Частное от деления (наиболее значащая цифра результата) помещается в регистр AH, а остаток (наименее значащая цифра результата) — в регистр AL .
Для того, чтобы преобразовать содержимое регистра AX к ASCII-формату, необходимо после команды AAM выполнить команду OR AX,0x3030h. Пример:

```
mov AL,3     ; множимое в формате неупакованного BCD помещается в регистр AL
mov BL,9     ; множитель в формате неупакованного BCD помещается в регистр BL
mul BL       ; AX = 0x03 * 0x09 = 0x001Bh
aam          ; AX = 0x0207h
or AX,3030H  ; AX = 0x3237h, т.е. AH = '2', AL = '7'
```

AAD imm8
Команда AAD (ASCII Adjust AX Before Division) используется для подготовки двух разрядов неупакованных BCD-цифр (наименее значащая цифра в регистре AL, наиболее значащая цифра в регистре AH) для операции деления DIV, которая возвращает неупакованный BCD-результат.
!!!!!!!Команда AAD устанавливает регистр AL в значение AL = AL + (imm8 * AH) , где imm8 – это второй байт кода операции ib (равный 0x0Ah для безоперандной мнемоники AAD), с последующей очисткой регистра AH. После команды AAD регистр AX будет равен двоичному эквиваленту оригинального неупакованного двухзначного числа.
Пример:

```
mov AX,0207H   ; делимое в формате неупакованного BCD помещается в регистр AX
mov BL,05H     ; делитель в формате неупакованного BCD помещается в регистр BL
aad            ; AX = 0x001Bh
div BL         ; AX = 0x0205h
or AL,30H      ; AL = 0x35h = '5'
```

13. Команды побитовых операций. Логические команды.

Операции над битами и байтами
BT <база>, <смещение>

Считывает в CF значение бита из битовой строчки.
Команда BT считывает в флаг CF значение бита из битовой строки, определенной первым операндом - битовой базой (регистр или переменная), со смещением, указанным во втором операнде - битовом смещении (число или регистр). Когда первый операнд - регистр, то битовой базой считается бит 0 в названном регистре и смещение не может превышать 15 или 31 (в зависимости от размера регистра); если оно превышает эти границы, в качестве смещения будет использоваться остаток от деления на 16 или 32 соответственно.
!!!!!!!!Если первый операнд - переменная, то в качестве битовой базы нужен бит 0 указанного байта в памяти, а смещение может принимать значения от 0 до 31, если оно установлено непосредственно (старшие биты процессором игнорируются), и от -231 до 231—1, если оно указано в регистре.

После выполнения команды BT флаг CF равен значению считанного бита, флаги OF, SF, ZF, AF и PF не определены.

BTS <база>, <смещение>

Устанавливает бит в 1.

BTR <база>, <смещение>

Сбрасывавет бит в 0.

BTC <база>, <смещение>

Инвертирует бит.

Эти три команды соответственно устанавливают в 1 (BTS), сбрасывают в 0 (BTK) и инвертируют (BTC) значение бита, который находится в битовой строке с началом, определенным в базе (регистр или переменная), и смещением, указанным во втором операнде (число от 0 до 31 или регистр). Если битовая база - регистр, то смещение не может превышать 15 или 31 в зависимости от разрядности этого регистра. Если битовая база - переменная в памяти, то смещение может принимать значения от —2^31 до 2^31—1 (при условии, что оно указано в регистре). После выполнения команд BTS, BTR и BTC флаг CF равен значению считанного бита до его изменения в результате действия команды, флаги OF, SF, ZF, AF и PF не определены.

BSF <приемник>, <смещение>

Прямой поиск бита (от младшего разряда).

BSR <приемник>, <смещение>

Обратный поиск бита (от старшего разряда).

BSF сканирует источник (регистр или переменная), начиная с самого младшего бита, и записывает в приемник (регистр) номер первого встретившегося бита, равного 1. Команда BSR сканирует источник, начиная с самого старшего бита, и возвращает номер первого встретившегося ненулевого бита, считая от нуля. То есть, если источник равен 0000 0000 0000 000b, то BSF возвратит 1, а BSR - 14. Если весь источник равен нулю, значение приемника не определено и флаг ZF устанавливается в 1, иначе ZF всегда сбрасывается. Флаги CF, OF, SF, AF и PF не определены.

SETcc <приемник>

Выставляет приемник (1 байт) в 1 или 0 в зависимости от условия, аналогично Jcc. Это набор команд, устанавливающих приемник (8-битный регистр или переменная размером в 1 байт) в 1 или 0, если удовлетворяется или не удовлетворяется определенное условие.

!!!!!!Фактически в каждом случае проверяется состояние тех или иных флагов, но, когда команда из набора SETcc используется сразу после CMP, условия приобретают формулировки, соответствующие отношениям между операндами CMP. Скажем, если операнды CMP были неравны, то команда SETNE, выполненная сразу после CMP, установит значение своего операнда в 1.

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).

При выполнении циклического сдвига на 1, команды ROR (ROL) перемещают каждый бит приемника вправо (влево) на одну позицию, за исключением самого младшего (старшего), который записывается в позицию самого старшего (младшего) бита

RCR (RCL)

Циклический сдвиг вправо (влево) через CF.

Команды RCR и RCL выполняют аналогичное действие, но включают флаг CF в цикл, как если бы он был дополнительным битом в приемнике

14. Команды работы со строками.

Строковые операции: копирование, сравнение, сканирование, чтение, запись
Строка-источник - DS:SI, строка-приёмник - ES:DI.
За один раз обрабатывается один байт (слово).

- MOVS / MOVSB / MOVSW <приёмник>, <источник> - копирование
- CMPS / CMPSB / CMPSW <приёмник>, <источник> - сравнение
- SCAS / SCASB / SCASW <приёмник> - сканирование (сравнение с AL/AX (в зависимости от размеров приемника)
- LODS / LODSB / LODSW <источник> - чтение (в AL/AX)
- STOS / STOSB / STOSW <приёмник> - запись (из AL/AX)
- 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.

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. Команда трансляции по таблице.

XLAT адрес Трансляция в соответствии с таблицей
XLATB
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.

16. Команда вычисления эффективного адреса.

LEA <приемник>,<источник>, назначение: Вычисление эффективного адреса
Вычисляет эффективный адрес источника (переменная) и помещает его в приемник (регистр). С помощью LEA можно вычислить адрес переменной, которая описана сложным методом адресации, например по базе с индексированием.
Если адрес - 32-битный, а регистр-приемник - 16-битный, старшая половина вычисленного адреса теряется, если наоборот, приемник - 32-битный, а адресация - 16-битная, то вычисленное смещение дополняется нулями.

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

Структура программы на ассемблере

- Модули (файлы исходного кода)
- Сегменты (описание блоков памяти)
- Составляющие программного кода:
    - команды процессора
    - инструкции описания структуры, выделения памяти, макроопределения
- Формат строки программы:
    - метка команда/директива операнды ; комментарий

Любая программа состоит из сегментов
Директива SEGMENT
Любая программа состоит из сегментов из одного или нескольких. Обычно область памяти, в которой находятся команды, называют сегментом кода, область памяти с данными - сегментом данных и область памяти, отведенную под стек, - сегментом стека.
/// ! Виды сегментов:

- Сегмент кода CS
- Сегмент данных DS
- Сегмент стека SS

/// ! Описание сегмента в исходном коде:
имя SEGMENT READONLY выравнивание тип разряд 'класс'
…
имя ENDS

- Выравнивание по умолчанию - параграф (PARA)
- Тип по умолчанию - PRIVATE
- Класс - любая метка, взятая в одинарные кавычки. Все сегменты с одинаковым классом, будут расположенным друг за другом (в исполняемом файле, даже PRIVATE)

Выравнивание:

- BYTE
- WORD
- DWORD
- PARA (по умолчанию)
- PAGE

Тип:

- PUBLIC - 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 - определение сегмента стека. Заставляет компоновщик соединить все одноименные сегменты и вычислять адреса в этих сегментах относительно регистра SS.

- COMMON - 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 - располагает сегмент по абсолютному адресу параграфа (параграф — объем памяти, кратный 16, поэтому последняя шестнадцатеричная цифра адреса параграфа равна 0). Все метки и адреса в определенном таким образом сегменте отсчитываются относительно заданного абсолютного адреса;
- PRIVATE (по умолчанию) - сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля.

Директива ASSUME
ASSUME регистр : имя сегмента

- Не является командой
- Нужна для контроля компилятором правильности обращения к переменным
- устанавливает значение сегментного регистра по умолчанию

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 начинается выполнение программы.
- Если в программе несколько модулей, только один может содержать начальный адрес.

18. Подпрограммы. Объявление, вызов.
Описание подпрограммы
имя_подпрограммы PROC [NEAR | FAR] ; по умолчанию NEAR, если не указать
;тело подпрограммы;
и в конце ставим ret [кол-во используемых локальный переменных] ;
если не использовались локальные переменные на стеке то тогда ничего не указывается
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)
- Передает 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

19. Подпрограммы. Возврат управления.

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).

RET Возврат из процедуры
RETN Возврат из ближней процедуры
RETF Возврат из дальней процедуры

Команда ret извлекает из стека адрес возврата и передает управление назад в программу, первоначально вызвавшую процедуру. Если командой ret завершается ближняя процедура, объявленная с атрибутом near, или используется модификация команды retn, тогда со стека снимается одно слово- относительный адрес точки возврата. Передача управления в этом случае осуществляется в пределах одного программного сегмента. Если командой ret завершается дальняя процедура, объявленная с атрибутом far, или используется модификация команды retf, со стека снимаются два слова: смещение и сегментный адрес точки возврата. В этом случае передача управления может быть межсегментной.
В команду ret может быть включен необязательный операнд (кратный 2), который указывает, на сколько байтов дополнительно смещается указатель стека после возврата в вызывающую программу. Прибавляя эту константу к новому значению SP, команда ret обходит аргументы, помещенные в стек вызывающей программой (для передачи процедуре) перед выполнением команды call. Обе разновидности команды не воздействуют на флаги процессора.

20. Макроопределения.

Макроопределение (макрос) - именованный участок программы, который ассемблируется каждый раз, когда его имя встречается в тексте программы.

Роль макросов в ассемблере такая же, как макросов в си. Очень гибкий и мощный инструмент, чтобы писать код общего вида, который во время работы препроцессора будет заменяться на конкретные выражения.

Определения:

1. Макроопределение - специальным образом оформленная последовательность предложений языка ассемблера, под управлением которой ассемблер (точнее, его часть, называемая макрогенератором или препроцессором) порождает макрорасширения макрокоманд.
2. Макрорасширение - последовательность предложений языка ассемблера (обыкновенных директив и команд), порождаемая макрогенератором при обработке макрокоманды под управлением макроопределения и вставляемая в исходный текст программы вместо макрокоманды.
3. Макрокоманда (или макровызов) - предложение в исходном тексте программы, которое воспринимается макрогенератором как команда (приказ), предписывающая построить макрорасширение и вставить его на ее место.

- В макрокоманде могут присутствовать параметры, если они были описаны в макроопределении.
- Макроопределение без параметров однозначно определяет текст макрорасширения.
- Макроопределение с параметрами описывает множество (возможно, очень большое) возможных макрорасширений, а параметры, указанные в макрокоманде, сужают это множество до одного единственного макрорасширения.

Определение макроса в программе:
имя MACRO параметры
...
ENDM

Пример:

```
load_reg MACRO register1, register2
push register1
pop register2
ENDM
```

Сравнение макросов с подпрограммами
Плюсы:

- Так как текст макрорасширения вставляется на место макрокоманды, то нет затрат времени, как для подпрограмм, на подготовку параметров, передачу управления и выполнение других работ при выполнении программы

Минусы:

- При многочисленных вызовах МО (макроопределения) разрастается объем модуля программы,
- Фактические значения параметров макрокоманд должны быть известны препроцессору или могли быть вычислены им (нельзя использовать в качестве фактического параметра МО значения переменных или регистров, так как они могут быть известны только при выполнении программы).

Замечания.

- Имена формальных параметров МО-й локализованы в них, т.е. вне определения могут использоваться для обозначения других объектов.
- Число формальных параметров ограничено лишь длиной строки, обрабатываемой ассемблером.
- МО-я должны предшествовать обращениям к ним.

- Нет ограничений, кроме физических, на число предложений в теле МО.
- В листинге предложениям макрорасширений предшествуют ЦБЗ, указывающие глубину их вложения в макроопределениях.

//
Директива присваивания =
Директива присваивания служит для создания целочисленной макропеременной или изменения ее значения и имеет формат:

Макроимя = Макровыражение

- Макровыражение (или Константное выражение) - выражение, вычисляемое препроцессором, которое может включать целочисленные константы, макроимена, вызовы макрофункций, знаки операций и круглые скобки, результатом вычисления которого является целое число
- Операции:
    - арифметические (+, -, *, /. MOD)
    - логические
    - сдвигов
    - отношения

Директивы отождествления EQU, TEXTEQU
Директива для представления текста и чисел:
Макроимя EQU нечисловой текст и не макроимя ЛИБО число
Макроимя EQU <Операнд>
Макроимя TEXTEQU Операнд

- % - вычисление выражение перед представлением числа в символьной форме (пример ниже - из методички)

- <> - подстановка текста без изменений (полезно, когда есть вероятность пересечения имени какого-либо макроса с простым (или не очень) текстом, который мы хотим вставить)
- & - склейка текста (A&B ==> AB, если параметры - макропараметры, то они склеятся)
- ! - считать следующий символ текстом, а не знаком операции (A!&B ==> A&B)
- ;; - исключение строки из макроса (После препроцессора эта строчка исчезнет (если одна ";", то комментарий остается); Дословно из методички: "текст не выносится в макрорасширение")

REPT
Повтор фиксированное число раз
REPT <число>
...
ENDM

IRP или FOR (конкретное имя зависит от компилятора)
Подстановка фактических параметров по списку на место формального
IRP form,<fact_1[,fact_2,...]>
...
ENDM

IRPC или FORC (конкретное имя зависит от компилятора)
Подстановка символов строки на место формального параметра

```
IRPC form,fact
...
ENDM
```

Стандартные макрофункция @SubStr и директива SubStr могут порождать множество подстрок типа text с числовыми и нечисловыми значениями, причём при одних и тех же значениях параметров директива SubStr определит (переопределит) макропеременную типа text, а макрофункция @SubStr вернёт значение, совпадающее со значением макропеременной. Следующий вложенный цикл позволяет перебрать и вывести значения подмножеств строки 1234

```
j=1
while j LE 4
        i=1
        WHILE i le 5-j
                names SubStr <ABCD>,i,j
                %ECHO 'names SubStr <ABCD>,i,j' out: names , i, j
                %ECHO '@SubStr (ABCD,i,j)' out: @SubStr (ABCD,i,j)
                i=i+1
        endM
        j=j+1
endm
```

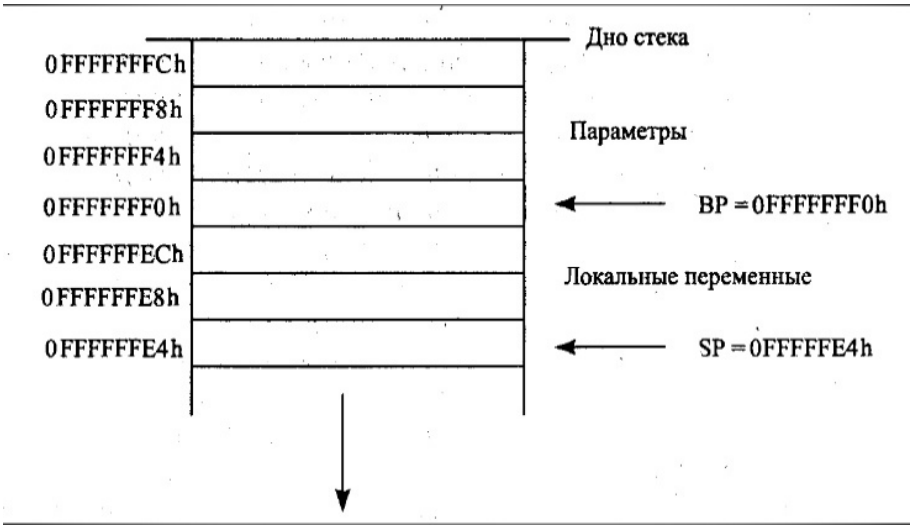
```
IF:
IF c1
…
ELSEIF c2
…
ELSE
…
ENDIF
```

- IFB <par> - истинно, если параметр не определён (то есть фактический параметр par не был задан в МКоманде)
- IFNB <par> - истинно, если параметр определён
- IFIDN <s1>,<s2> - истинно, если строки совпадают
- IFDIF <s1>,<s2> - истинно, если строки разные
- IFDEF/IFNDEF <name> - истинно, если имя объявлено/не объявлено

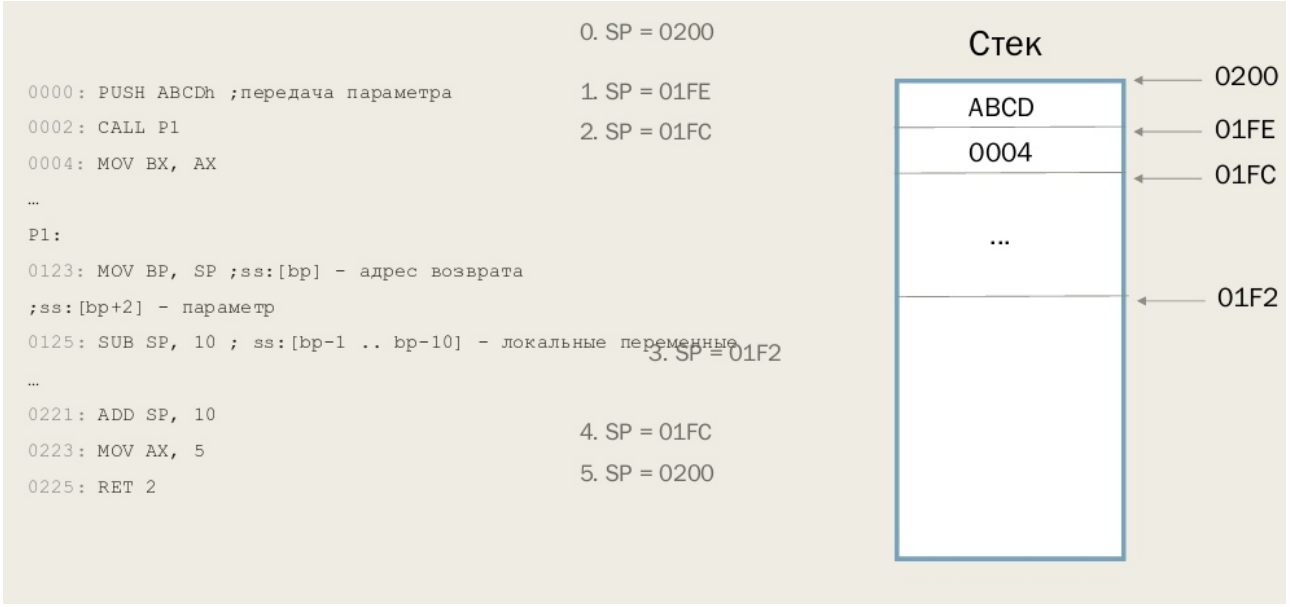
21. Стек. Аппаратная поддержка вызова подпрограмм.

Стек. Назначение, примеры использования.
Стек работает по правилу LIFO / FILO (последним пришёл, последним вышел)
Сегмент стека — область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний.
Используется для временного хранения переменных, передачи параметров для подпрограм, адрес возврата при вызове процедур и прерываний.
Регистр SP — указывает на вершину стека
В x86 стек "растёт 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)
Участие стека в механизме вызова подпрограммы и возврата из нее является решающим. Поскольку в стеке хранится адрес возврата, подпрограмма, сама используя стек, например, для хранения промежуточных результатов, обязана к моменту выполнения команды ret вернуть стек в исходное состояние. Команда ret никак не анализирует состояние или содержимое стека. Она просто снимает со стека верхнее слово, считая его адресом возврата, и загружает это слово в указатель команд IP. Если к моменту выполнения команды ret указатель стека окажется смещенным в ту или иную сторону, команда ret по-прежнему будет рассматривать верхнее слово стека, как адрес возврата, и передаст по нему управление, что неминуемо приведет к краху системы.



Команды работы со стеком
PUSH <источник> — поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
POP <приемник> — 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. (регистры общего назначения + SP + BP)
POPA — загрузить 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)
Примеры использования



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

Таблица векторов прерываний в реальном режиме работы процессора
● 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) - 6 байт
- 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))

Перехват прерывания

● Сохранение адреса старого обработчика
● Изменение вектора на "свой" адрес
● Вызов старого обработчика до/после отработки своего кода
● При деактивации - восстановление адреса старого обработчика

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

Префикс программного сегмента - структура данных, которая используется в DOS для сохранения состояния программы, располагается в начале программы и занимает 256 байт

ИЗ МОЕЙ ЛАБЫ ТЕОРИЯ
В точку init передается управление при запуске программы.
ЕЕ основная задача — установить резидент в памяти
Для замены вектора прерывания на свой адрес используем функцию 35h, номер прерывания - 08 - таймер
затем сохраняем указатели на программу обработки (смещение сегментного обработчика). В итоге в old хранится старый обработчик 08h
Устанавливаем наш обработчик
Для замены вектора прерывания на свой адрес используем функцию 25h, номер прерывания - 08 - таймер
При вызове функции 25h в регистр AL помещается номер модифицируемого вектора, а в регистры DS:DX - полный двухсловный адрес нового обработчика.
завершаем программу с сохранением резидентной части в оперативке (31 - команда, 00 - код завершения)

В резидентной программе
сохраняем содержимое регистров флагов FLAGS
вызываем старый обработчик
пушим в стек чтобы по завершению обработчика прерывания все регистры вернулись в свое первоначальное состояние
Зануляем ах и кидаем туда таймер. Увеличиваем на 1
Если таймер != 18, то завершаем обработчик(он потом вывозится снова) и чистим стек
Обнуляем таймер. Если скорость != 0, то мы ее уменьшаем на 1, тк прыгаем от быстрой скорости к маленькой

Для установки характеристик режима автоповтора в порт 60h необходимо записать код команды 0F3h, затем байт, определяющий характеристики режима

(команда F3h отвечает за параметры режима автоповтора нажатия клавиши.)

Порт 60h доступен для записи и обычно принимает пары байтов последовательно: первый - код команды, второй - данные. OUT - команда записи в порт вывода

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 со старым 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. 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.

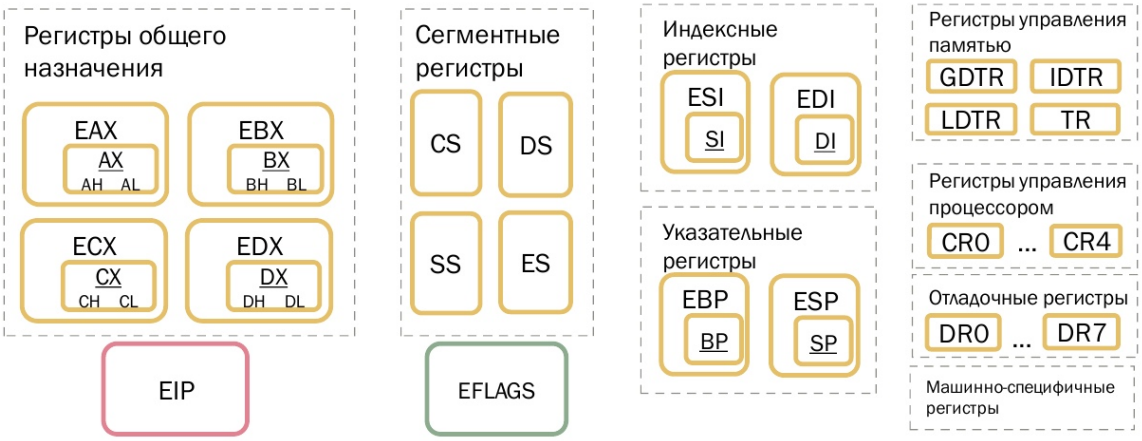
32-разрядные:

- 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.

Сопроцессор (FPU – Floating Point Unit)

Отдельное опциональное устройство на материнской плате, встроен в процессор.

FPU — блок, производящий операции с плавающей точкой или математический сопроцессор.

FPU помогает основному процессору выполнять математические операции над вещественными числами.

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).

Представление вещественных чисел

- Нормализованная форма представления числа (1,...*2^exp);
- Экспонента увеличена на константу для хранения в положительном виде;
- Пример представления 0,625 в коротком вещественном типе:
  - 1/2 + 1/8 = 0,101b;
  - 1,01b*2^-1;
  - Бит 31 - знак мантиссы, 30-23 - экспонента, увеличенная на 127, 22-0 - мантисса без первой цифры;
  - 0 01111110 01000000000000000000000.
- Все вычисления FPU - в расширенном 80-битном формате.

Особые числа FPU

■ Положительная бесконечность: знаковый - 0, мантисса - нули, экспонента - единицы

■ Отрицательная бесконечность: знаковый - 1, мантисса - нули, экспонента - единицы

■ NaN (Not a Number): – qNAN (quiet) - при приведении типов/отдельных сравнениях – sNAN (signal) - переполнение в большую/меньшую сторону, прочие ошибочные ситуации

■ Денормализованные числа (экспонента = 0): находятся ближе к нулю, чем наименьшее представимое нормальное число.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Бесконечности возникают при делении бесконечности (или числа) на ноль.
NaN делятся на тихие и сигнальные.
Тихий - не приводит к исключению (арифмитической ошибки нет, но получить число без округления нельзя)
Сигнальный - переполнения в большую меньшую сторону (при делении на ноль иногда)
Денормализованные числа - такие, которые не укладываются в заданный формат представления числа и позволяют хранить еще меньшие числа (в экспоненте все нули - специальное значение), мантисса считается умноженной на 2 в отрицательной степени, которая еще меньше, чем минимальное значение экспоненты. Перевод в денормализованные числа может производится аппаратно при получении очень малых значений. Их обработка может производится дольше, чем обработка нормализованных чисел.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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

FPU — блок, производящий операции с плавающей точкой или математический сопроцессор.
FPU помогает основному процессору выполнять математические операции над вещественными числами.

1. Команды пересылки данных:
■ FLD - загрузить вещественное число из источника (переменная или ST(n)) в стек. Номер вершины в SR увеличивается
■ FST/FSTP - скопировать/считать число с вершины стека в приёмник
■ FILD - преобразовать целое число из источника в вещественное и загрузить в стек
■ FIST/FISTP - преобразовать вершину в целое и скопировать/считать в приёмник
■ FBLD, FBSTP - загрузить/считать десятичное BCD-число
■ FXCH - обменять местами два регистра (вершину и источник) стека
2. Арифметические команды:
■ FADD, FADDP, FIADD - сложение, сложение с выталкиванием из стека, сложение целых. Один из операндов - вершина стека
■ FSUB, FSUBP, FISUB - вычитание
■ FSUBR, FSUBRP, FISUBR - обратное вычитание (приёмника из источника)

■ FMUL, FMULP, FIMUL - умножение
■ FDIV, FDIVP, FIDIV - деление
■ FDIVR, FDIVRP, FIDIVR - обратное деление (источника на приёмник)
■ FPREM - найти частичный остаток от деления (делится ST(0) на ST(1)). Остаток ищется цепочкой вычитаний, до 64 раз
■ FABS - взять модуль числа
■ FCHS - изменить знак
■ FRNDINT - округлить до целого
■ FSCALE - масштабировать по степеням двойки (ST(0) умножается на 2ST(1))
■ FXTRACT - извлечь мантиссу и экспоненту. ST(0) разделяется на мантиссу и экспоненту, мантисса дописывается на вершину стека
■ FSQRT - вычисляет квадратный корень ST(0)
3. Команды сравнений:
■ FCOM, FCOMP, FCOMPP - сравнить и вытолкнуть из стека
■ FUCOM, FUCOMP, FUCOMPP - сравнить без учёта порядков и вытолкнуть
■ FICOM, FICOMP, FICOMP - сравнить целые
■ FCOMI, FCOMIP, FUCOMI, FUCOMIP (P6)
■ FTST - сравнивает с нулём
■ FXAM - выставляет флаги в соответствии с типом числа

4. Трансцедентные операции сопроцессора:
■ FSIN
■ FCOS
■ FSINCOS
■ FPTAN
■ FPATAN
■ F2XM1 – 2 x -1
■ FYL2X, FYL2XP1 – y*log2 x, y*log2 (x+1)
5. Константы FPU:
■ FLD1 – 1,0
■ FLDZ - +0,0
■ FLDPI - число Пи
■ FLDL2E - log2e
■ FLDL2T - log210
■ FLDLN2 – ln(2)
■ FLDLG2 – lg(2)
6. Команды управления:
■ FINCSTP, FDECSTP - увеличить/уменьшить указатель вершины стека
■ FFREE - освободить регистр
■ FINIT, FNINIT - инициализировать сопроцессор / инициализировать без ожидания (очистка данных, инициализация CR и SR по умолчанию)
■ FCLEX, FNCLEX - обнулить флаги исключений / обнулить без ожидания
■ FSTCW, FNSTCW - сохранить CR в переменную / сохранить без ожидания
■ FLDCW - загрузить CR
■ FSTENV, FNSTENV – сохранить вспомогательные регистры (14/28 байт) / сохранить без ожидания
■ FLDENV - загрузить вспомогательные регистры
■ FSAVE, FNSAVE, FXSAVE - сохранить состояние (94/108 байт) и инициализировать, аналогично FINIT
■ FRSTOR, FXRSTOR - восстановить состояние FPU
■ FSTSW, FNSTSW - сохранение CR
■ WAIT, FWAIT - обработка исключений

■ FNOP - отсутствие операции

Исключения

- Неточный результат - произошло округление по правилам, заданным в CR. Бит в SR хранит направление округления
- Антипереполнение - переход в денормализованное число
- Переполнение - переход в "бесконечность" соответствующего знака
- Деление на ноль - переход в "бесконечность" соответствующего знака
- Денормализованный операнд
- Недействительная операция

27. Расширения процессора. MMX. Регистры, поддерживаемые типы данных.
MMX - Расширение, которое было встроено для увеличения эффективности обработки больших потоков данных (простые операции над массивами однотипных данных (звук, изображения, видеопоток))

- 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 потому что они используют одни регистры и 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))

MMX-регистры — это восемь 64-разрядных регистров (MM0 … MM7), которые отображены на младшие 64 бита регистров общего назначения FPU (R0 … R7)(регистры, которые рассматриваются в качестве стека). То есть регистры MMX по сути являются логическими регистрами, физически совмещенными с регистрами R0 … R7. Это значит, что при записи информации в MMX-регистр, записываемое значение автоматически появляется в младших битах соответствующего регистра FPU, и наоборот, при записи в регистровый стек FPU значения младших 64 бит записываемой величины окажутся в соответствующем MMX-регистре. Поскольку регистры FPU (R0 … R7) организуют так называемый регистровый стек (ST0 … ST7), то в зависимости от текущего значения поля TOP регистра состояния FPU (SW) будет меняться соответствие регистров стека и MMX-регистров.

Типы данных MMX:

- учетверенное 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

28. Расширения процессора. MMX. Классификация команд.

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))

Классификация команд:
1. Команды пересылки данных:
   - MOVD/MOVQ - пересылка двойных/учетверенных слов;
   - PACKSSWB - (байты и слова зависит от последней буквы) упаковка со знаковым насыщением слов (приемник -> младшая половина приемника, источник -> старшая половина приемника, в случае наличия значащих разрядов в отбрасываемых частях происходит насыщение);
   - PACKUSWB - упаковка слов с беззнаковым насыщением, распаковка и объединение старших элементов источника и приемника через 1
2. Арифметические операции:
   - поэлментное сложение (перенос игнорируется, побайтовое сложение)
   - PADDSB/PSUBSB - сложение/вычитание с насыщением
   - PADDUSB - беззнаковое сложение с насыщением
   - PSUBB - вычитание (заем игнорируется)
   - PSUBSB - вычитание с насыщением
   - PMULHW/PMULLW - старшое/младшее умножение (сохраняет старшую или младшую части результата в приемник)
   - PMADDWD - умножение и сложение (перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших)
3. Команды сравнения:
   - PCMPEQB - (байты и слова и двойное слово зависит от последней буквы) проверка на равенство (Если пара равна - соответствующий элемент приёмника заполняется единицами, иначе - нулями)
   - PCMPGTB - проверка на больше (Если элемент приёмника больше, то заполняется единицами, иначе - нулями)
4. Логические операции:
   - PAND - логическое И
   - POR - логическое ИЛИ
   - PANDN - логическое НЕ-И (штрих шеффера)
   - PXOR - Логическое исключающее ИЛИ
5. Сдвиговые операции:
   - PSLLW - логический влево (слово/двойное слово/во всем регистре зависит от последней буквы)
   - PSRLW - логический вправо(слово/двойное слово/во всем регистре зависит от последней буквы)
   - PSRAW - арифметический вправо(слово/двойное слово зависит от последней буквы)

29. Расширения процессора. SSE. Регистры, поддерживаемые типы данных.
30. Расширения процессора. SSE. Классификация команд.

Расширение предназначается для современных приложений, работающих с двумерной и трехмерной графикой, видео-, аудио- и другими видами потоковых данных. В отличие от MMX, это расширение не использует уже существующие ресурсы процессора, а вводит 8 новых независимых 128-битных регистров данных: XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6 и XMM7. Таким образом решаются проблемы технологии MMX - не требуется команд типа EMMS для переключения режимов и можно пользоваться другими расширениями, работая с SSE.
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

Типы данных SSE
Основной тип данных, с которым работают команды SSE, -упакованные числа с плавающей запятой одинарной точности. В одном 128-битном регистре размещаются сразу четыре таких числа - в битах 127-96 (число 3), 95-64 (число 2), 63-32 (число 1) и 31-0 (число 0). Это стандартные 32-битные числа с плавающей запятой, используемые числовым сопроцессором. Целочисленные команды SSE могут работать с упакованными байтами, словами или двойными словами. Однако эти команды оперируют данными, находящимися в регистрах MMX.

ВТОРОЙ ВОПРОС

Классификация команд:

1. Команды пересылки данных:

- MOVAPS - Переслать выравненные упакованные числа
- MOVUPS - Переслать невыравненные упакованные числа
- MOVHPS/MOVLPS - Переслать старшие/младшие упакованные числа
- MOVHLPS/MOVLHPS - Переслать старшие упакованные числа в младшие, Переслать младшие упакованные числа в старшие

2. Арифметические операции:

- ADDPS - Сложение упакованных вещественных чисел
- ADDSS - Сложение одного вещественного числа
- SUBPS - Вычитание упакованных вещественных чисел
- SUBSS - Вычитание одного вещественного числа
- MULPS - Умножение упакованных вещественных чисел
- MULSS - Умножение одного вещественного числа
- DIVPS - Деление упакованных вещественных чисел
- DIVSS - Деление одного вещественного числа
- SQRTPS - Корень из упакованных вещественных чисел
- SQRTSS - Корень из одного вещественного числа

3. Команды сравнения:

- CMPPS - Сравнение одной пары упакованных чисел
- UCOMISS - Сравнение одной пары неупорядоченных чисел с установкой флагов

4. Логические операции:

- ANDPS - логическое И
- ORPS - логическое ИЛИ
- ANDNPS - логическое НЕ-И (штрих шеффера)
- XORPS - Логическое исключающее ИЛИ

5. Сдвиговые операции:

- логический влево
- логический вправо
- арифметический вправо

FPU — блок, производящий операции с плавающей точкой или математический сопроцессор.
FPU помогает основному процессору выполнять математические операции над вещественными числами.

MMX - набор инструкций, позволяющих выполнять характерные для процессов кодирования и декодирования потоковых аудио/видео данных действия за одну машинную инструкцию

SSE - расширение инструкций процессора для потоковой обработки в режиме SIMD, т.е. когда требуется применять однотипные операции к потоку данных.

Технология SSE позволяла преодолеть 2 основные проблемы MMX — при использовании MMX невозможно было одновременно использовать инструкции сопроцессора, так как его регистры были общими с регистрами MMX, и возможность MMX работать только с целыми числами.

MMX - использует регистры сопроцессора ST обозначаемые MMX для реализации целочисленных операций.
Размер регистра MMX=64бита
SSE, SSE2, .. - использует уже собственные регистры XMM
Размер регистра XMM=128бит

MMX, SSE отличаются прежде всего наборами команд и типами данных