1. Архитектура фон Неймана, принципы фон Неймана
Процессор состоит из блоков **УУ** и **АЛУ**
УУ (управляющее устройство) - дискретный конечный автомат. Структурно состоит из: дешифратора команд (операций), регистра команд, узла вычислений текущего исполнительного адреса, счетчика команд (**регистр IP**). **АЛУ** (арифметическое лог устройство) - 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. решение задач через програ
3.Принцип однородности памяти. Память используется не только для хранения данных, но и для программ.
4.Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы.
5. Возможность условного перехода в процессе выполнения программы. в процессе выполнения одной прог переход к другo память ⇔ Процессор (УУ АЛУ) {in ввод out вывод}
Архитектура фон Неймана - принцип совместного хранения команд и данных в одной памяти компьютера.

2. Маш команд , маш код. Понятие языка ассемблера.
Машинная команда - инструкция (в двоичном коде) из аппаратно определённого набора, которую способен выполнять процессор. **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. слово - 2 байта двойное слово 4байта 4ное слово - 8 байта Параграф - 16 байт . 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. **регистры проц Внешняя память** - диски, флешки и т.д.
Исполняемый файл - файл, содержащий прогу в виде, в котором она может быть исполнена компом (маш код)
Получение исполняемых файлов
● Компилятор - программа для преобразования исходного текста другой проги на опред языке в объектный модуль.
● Компоновщик (линковщик) - программа для связывания нескольких объектных файлов в исполняемый .EXE и .COM
Послед запуска программы операционной системой:
1. Определение формата файла.
2. Чтение и разбор заголовка.
3. Считывание разделов исполняемого модуля (файла) в ОЗУ по необходимым адресам.
4. Подготовка к запуску, если требуется (загрузка библиотек).
5. Передача управления на точку входа.
Исполнение программы
1. Передача управления на точку входа.
2. Выполнение инструкций по одной.
3. Передача управления на следующую инструкцию.
4. Повторение п.2-3 до тех пор, пока не будет достигнута инструкция завершения программы.
5. Остановка программы.
6. Освобождение памяти, занятой программой.
7. Возврат управления операционной системе.

4. Сегментная модель памяти в архитектуре 8086
Сегментная модель памяти - модель памяти, в которой для формирования адреса используются 2 числа: адрес начала сегмента и смещение искомого байта относительно этого начала. Программы могут работать с памятью как с несколькими непрерывными массивами
Сегмент - непрерывный участок памяти, начинающийся с адреса, кратного 16. Минимальный размер сегмента 16 байт. Максимальный размер сегмента 64 Кбайт.
Шина адреса - 20 бит, что позволяет адресовать 2^20 (1мб) памяти, а не 2^16 б. **Шина данных** - 16 бит
Плоская модель памяти - модель памяти flat (бессегментная); программы могут работать с памятью как с одним непрерывным массивом (адреса начала всех сегментов равны).

5. Процессор 8086. Регистры общего назначения
1980-85 годах Разрядность: 16 бит. Всего в 8086 14 регистров. имеется 14 16-разрядных регистров:
Регистр **процессора** — блок ячеек памяти, образующий сверхбыструю оперативную память (СОЗУ) внутри процессора;
регистров общего назначения (AX, BX, CX, DX), Каждый из регистров имеет старшую и младшую часть, по 1 байту на каждый. **AL** (младшая часть) и **AH** (старшая часть).AX (Accumulator): используется в операциях умножения, деления, обмена с устройствами ввода-вывода **BX** (*база*) - используется в качестве указателя на данные в памяти. **CX** (*счетчик*) - используется в операциях циклического счета.
DX (*data*) - используется в операциях ввода-вывода. Также может содержать данные, передаваемые для обработки в подпрограммах. 2 индексных регистра (**SI, DI**), 2 указательных (**BP, SP**), 4 сегментных регистра (**CS, SS, DS, ES**), программный счётчик или указатель на след команду (**IP**) и регистр флагов (**FLAGS**, включает в себя 9 флагов).

6. Процессор 8086. Сегментные регистры. Адресация в реальном
Каждый **сегментный регистр** определяет адрес начала сегмента в памяти, при этом сегменты могут накладывать или пересекаться. Сегменты способны оказаться где угодно, программа обращается к ним, применяя вместо полного адреса начала сегмента - 16-битный **селектор сегмента,** который хранится в соответствующем регистре сегмента.
CS - Code Segment - используется при выборке инструкций **SS** - Stack Segment - используется при работе со стеком **DS** и **ES** - Data Segment и Extra Segment - используются при обращении к данным
Сегментная часть адреса - адрес начала сегмента, который хранится в соответствующем регистре сегмента.
Смещение - адрес байта относительно начала сегмента.
Адресация в реальном режиме
В реальном режиме виртуальный адрес ячейки памяти состоит из двух чисел: **1 сегментной части** разрядностью 16 бит, по которой вычисляется физический адрес начала сегмента;
2 смещения разрядностью 16 бит ячейки памяти от начала сегмента.
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 к нему смещение.
[SEG]:[OFFSET] => физический адрес : [SEG] * 16 + [OFFSET]
Например, логический адрес 7522:F139 дает физический адрес 84359. На адресную шину передается именно физический адрес. Если результат больше, чем 2^20-1, то 21ый бит отбрасывается. 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.
Сегментный регистр хранит в себе старшие 16 разрядов (из 20) адреса начала сегмента. 4 младших разряда в адресе начала сегмента всегда нулевые. Говорят, что сегментный регистр содержит в себе номер параграфа начала сегмента Вычисление физического адреса выполняется процессором аппаратно, без участия прог.

7. Процессор 8086. Регистр флагов.
Флаги - выставляются при выполнении операций, в основном арифметических. Хотя разрядность регистра FLAGS 16 бит, реально используют не все 16. Остальные были зарезервированы при разработке процессора, но так и не были использованы Некоторые флаги напрямую менять нельзя, они меняются сами

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

TF, IF, IOPL, NT - системные флаги
CF (carry flag) - флаг переноса (при выходе за разрядную сетку)
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 1 если рез был равен **0**
SF (sign flag) - флаг знака равен старшему биту результату
TF (trap flag) - флаг трассировки для отладчика с пошаговым
IF (interrupt enable flag) - флаг разрешения прерываний если 0 процессор перестает обрабатывать прерывания от внешних устройств.
DF (direction flag) - флаг направления 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 flag) (появился в 286 процессоре) - уровень приоритета ввода-вывода
NT (nested task) (появился в 286 процессоре) - флаг вложенности задач
STC - устанавливает флаг CF в 1. **CLC** - устанавливает флаг CF в 0. **STD** - устанавливает флаг DF в 1. **CLD** - устанавливает флаг DF в 0. **STI** - устанавливает флаг IF в 1. **CLI** - устанавливает флаг IF в 0. **pushf** - помещает в стек значен регистра флагов. **popf** - извлекает из стека значен регистра флагов.

8. Команды пересылки данных.
MOV <приёмник>, <источник> Источник: непосредственный операнд (константа, включена в машинный код), РОН, сегментный регистр, переменная (ячейка памяти). Приёмник: РОН, сегментный регистр, переменная (ячейка памяти).
Ограничения
нельзя записывать данные в регистры CS и IP
нельзя копировать данные из одного сегментного регистра в другой сегментный регистр (сначала нужно скопировать данные в регистр общего назначения),
нельзя копировать непосредственное значение в сегментный регистр (сначала нужно скопировать данные в регистр общего назначения)
переменные не могут быть источником и приемником одновременно
МОЖНО mov ax, 5 mov bx, dx mov [1234h], ch mov ds, ax
НЕЛЬЗЯ: mov [0123h], [2345h] mov ds, 1000h
XCHG <операнд1>, <операнд2>Обмен операндов между собой. Выполняется либо над двумя регистрами, либо регистр

13. Команды побитовых операций. Логические команды
AND <приёмник>, <источник> - побитовое "И"
OR <приёмник>, <источник> - побитовое "ИЛИ"
XOR <приёмник>, <источник> - побито исключающее "ИЛИ" выстое занулен
NOT <приёмник> - инверсия
TEST <приемник>, <источник> Аналог AND, но результат не сохраняется. Выставляются флаги SF, ZF, PF.
SHL <приёмник>, <счётчик> - сдвиг влево (SAL - эквивал коман) -8>>2 = -4
SHR <приёмник>, <счётчик> - сдвиг вправо зануляет знак
SAR <приёмник>, <счётчик> - сдвиг вправо с сохранением знакового бита
ROR <приёмник>, <счётчик> - циклический сдвиг вправо
RCR <приёмник>, <счётчик> - цикл сдвиг вправо, через флаг переноса
ROL <приёмник>, <счётчик> - циклический сдвиг влево
RCL <приёмник>, <счётчик> - циклический сдвиг влево, через флаг
Все эти команды меняют регистр FLAGS

14. Команды работы со строками.
Есть 2 основных способа хранения текстовых строк в памяти компьютера:
Сишный: строка неограниченной длины, завершающаяся нулевым байтом
Паскалевский: в первых 1 или 2 байтах хранится длина строки, а затем идут сами символы.
Строка-источник - DS:SI, **строка-приёмник** - ES:DI. За один раз обрабатывается один байт (слово).
MOVS/MOVSB/MOVSW <приёмник>, <источник>
Копирует байт или слово из приемника в источник. После выполнения команды, регистры SI и DI увеличиваются на 1 (или 2), если флаг DF = 0, или уменьшаются на 1 (или 2), если DF = 1.
CMPS/CMPSB/CMPSW <приёмник>, <источник>
Сравнивает байт или слово из приемника с источником. После выполнения команды, регистры SI и DI также и выставление F
SCAS/SCASB/SCASW <приёмник>
сканирование источника (всегда это DS:SI), сравнение с AL/AX и выставление флага по его результатам, регистры SI и DI также
LODS/LODSB/LODSW <источник>
Чтение (в AL/AX). После выполнения команды, регистры SI и DI
STOS/STOSB/STOSW <приёмник>
Запись (из AL/AX). После выполнения команды, регистры также
В каждой команде источник и приемник можно опустить.
MOV DI, destination_string ; Загрузка адреса строки в регистр DI
MOV AL, 'A' ; Загрузка символа 'A' в регистр AL
WriteChar:
STOS BYTE PTR [DI] ; Запись символа в строку
INC DI ; Увеличение значения регистра DI на 1
JMP WriteChar ; Повторение операции
destination_string DB 255 DUP(0) ; Строка для записи символов

Эти команды работают только с отдельными байтами, машинными словами или двойными словами. Чтобы работать со всей строкой, нужно использовать префиксы.
REP - повторить следующую строковую операцию
REPE - повторить следующую строковую операцию, есливарно
REPZ - Повторить следующую строковую операцию, если нуль
REPNE - повторить след строковую операцию, если не равно
REPNZ - повторить след строковую операцию, если не нуль
Префиксы REP, REPE и REPNE применяются со строковыми операциями. Каждый префикс заставляет строковую команду, которая следует за ним, повторяться указанное в регистре счетчике (E)CX количество раз или, кроме этого, (для префиксов REPE и REPNE) пока не встретится указанное условие во флаге ZF. Пример использования: **REP LODS AX**

15. Команда трансляции по таблице
XLAT/XLATB - трансляция в соответствии с таблицей
XLAT [адрес] (принято писать адрес, но использоваться будет только сегментная часть, которая может собой заменять DS)
XLATB Помещает в AL байт из таблицы (массив, который не превышает 256 ячеек памяти) по адресу DS:BX со смещением относительно начала таблицы, равным AL. , XLATB -> AL = DS:[(E)BX + AL] Адрес, указанный в исходном коде, не обрабатывается компилятором и служит в качестве комментария. Если в адресе явно указан сегментный регистр, он будет использоваться вместо DS
Использование: Транслитерация (перевод английских символов в русские один к одному) Перевод шестнадцатеричных чисел в символы
По сути это аппаратная усечённая поддержка словарей.
Отличие XLAT от XLATB
mov bx, 0 ; поместит в bx адрес таблицы
mov al, 6 ; поместит в al номер байта в таблице
xlat ds:[bx+al] ; провели трансляцию - результат в al
Если мы не хотим указывать сегментный регистр, то можно использовать команду XLATB:
mov bx, 0 ; поместит в bx адрес таблицы
mov al, 6 ; поместит в al номер байта в таблице
xlatb ; провели трансляцию - результат в al
Пример: перевод десятичного числа (от 0 до 15) в соотв 16/с/с
; сегмент данных
table db '0123456789ABCDEF' ; таблица перекодировки
; сегмента кода
lea bx, table ; поместить в bx адрес таблицы
mov al, decimal_digit ; поместить в al десятичную цифру
xlatb

16. Команда вычисления эффективного адреса
LEA <приёмник>, <источник> Вычисляет эффективный адрес источника и помещает его в приёмник. Позволяет вычислить адрес, описанный сложным методом адресации. Поддерживает на аппаратном уровне (на уровне процессора) сложные формулы с несколькими операндами: сложение с константой и сложение/умножение каких-то регистров. Работает быстрее арифметических команд и не меняет флаги. 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. **Использование**: Вычисление адреса "на лету" Быстрая арифметика (не нужно писать несколько арифметических команд)

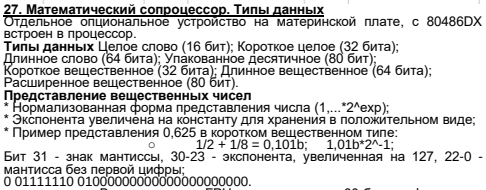
17. Структура программы на языке ассемблера. Модули. Сегменты.
Структура программы
1. модули (файлы исходного кода) - если в программе несколько модулей, только один может содержать начальный адрес.
2. сегменты (описание блоков памяти)
3.составляющие программного кода
команды процессора
инструкции описания структуры, выделения памяти, макроопределения
4. формат строки программы - строки имеют следующий вид: метка команда/директива операнды ; комментарий
Метка может быть любой комбинацией букв английского алфавита, цифр и символов (_ , ?, @, $), но не может начинаться с цифры. Метка может быть пустой. Поле команды может содержать команду процессора, которая транслируется в исполняемый код, или директиву, которая не приводит к появлению кода и управляет работой самого ассемблера.
Поле **операндов** содержит требуемые командой или директивой операнды. То есть нельзя указать операнды, но не указать команду или директиву. Поле **комментария** начинается с символа ; и продолжается до конца строки. Может быть пустым. Поле комментария не анализируется ассемблеро
Любая программа состоит из сегментов 1. **Сегмент кода** - содержит команды процессора, которые выполняются при работе программы. 2 **Сегмент данных** - содержит данные, с которыми работает программа. 3 **Сегмент стека** - содержит данные, используемые для организации стека. Сегмент программы описывается директивой **SEGMENT** и заканчивается директивой **ENDS**. Все пять операндов директивы SEGMENT являются необязательными. имя_сегмента …
segment readonly выравнивание тип разряд 'класс' …
имя_сегмента ends **имя_сегмента**
- метка, которая будет использоваться для получения сегментного адреса, а так же для комбинирования сегментов в группы.
readonly - указывает ассемблеру и компоновщику, с какого адреса может начинаться сегмент. Значения операнда:
byte - с любого адреса / word - с четного адреса dword - с адреса, кратного 4
para - с адреса, кратного 16 (значение по умолчанию) page - с адреса, кратного 256 **тип** -
определяет один из возможных типов комбинирования сегментов:
public - все такие сегменты с одинаковым именем, но разными классами объединяются в один сегмент **private** - сегмент такого типа не объединяется с другими сегментами **stack** - то же самое, что и public, но должен использоваться для сегмента стека, поэтому что при загрузке программы, стек получается как объединение всех сегментов типа stack **common** - сегменты с одинаковым именем также объединяются в один, но не последовательно, а по одному и тому же адресу. Следовательно, длина суммарного сегмента будет равна не сумме длин объединенных сегментов, а длине максимального. Можно формировать оверлейные программы **at** - выражение указывает, что сегмент должен располагаться по фиксированному абсолютному адресу в памяти. Результат выражения, используемого в качестве операнда для at, равен этому адресу, деленному на 16. Например, segment at 40h - сегмент начинается по абсолютному адресу 0400h. Такие сегменты обычно содержат только метки, указывающие на области памяти, которые могут понадобиться программе.
разряд - определяет размерность сегмента. Возможные значения: byte - размерность сегмента - 1 байт/ word - размерность сегмента - 2 байта dword - размерность сегмента - 4 байта / para - размерность сегмента - 16 байт / page - размерность сегмента - 256 байт / stack - размерность сегмента - 2 байта. **имя** сегмента - 0 байт **класс** - любая метка, взятая в одинарные кавычки. Сегменты одного класса располагаются в памяти последовательно. Если класс не указан, то используется класс DATA.
Директива ASSUME *регистр: имя сегмента* * Не является командой
* Нужна для контроля компилятором правильности обращения к переменным
Модель памяти .model модель, язык, модификатор TINY - один сегмент на всё **SMALL** - код в одном сегменте, данные и стек - в другом **COMPACT** - допустимо несколько сегментов данных **MEDIUM** - код в нескольких сегментах, данные - в одном **LARGE, HUGE** **Язык** - C, PASCAL, BASIC, SYSCALL, STDCALL. Для связывания с ЯВУ и вызова подпрограмм. **Модификатор** - NEARSTACK/FARSTACK Определение модели позволяет использовать сокращённые формы директив определения сегментов. Конец программы и точка входа … END start start - имя метки, объявленной в сегменте кода и указывающее на команду, с которой начнётся исполнение программы. Если в программе несколько модулей, только один может содержать начальный адрес.

24. Прерывания. Обработка прерываний в реал режиме работы процес
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
● 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
IRET используется для выхода из обработчика прерывания. Восстанавливает FLAGS, CS:IP. При необходимости выставить значение флага обработчик меняет его значение непосредственно в стеке
Срабатывание прерываний 1 Сохранение в текущий стек регистра флагов и адреса возврата (адреса следующей команды)
2 Передача управления по адресу обработчика из таблицы векторов
3.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 переназначивают стек) **4** 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))

25. Процессор 80386. Режимы работы. Регистры
32-разрядные:
- регистры, кроме сегментных
- шина данных
- шина адреса (2^32 = 4ГБ ОЗУ)

32-битный процессор 80386 имеет несколько режимов работы:
реальный режим (Real Mode) - совместимость с 8086.
защищенный режим (Protected Mode) * обращение к оперативной памяти по виртуальным адресам с использованием защиты памяти * набор доступных операций определяется уровнем привилегий (всего 4 кольца защиты): системный и пользовательский
виртуальный режим (Virtual Mode) - позволяет выполнение нескольких задач в реальном режиме на компьютерах, работающих в защищенном режиме. Виртуальный режим позволяет запускать программы для 8086 внутри защищенной среды. **Регистры**
EDX = Extended DX (обращение к частям остается (DX, DL))
Добавлены регистры поддержки работы в защищенном режиме (обеспечивание разделения доступа программ между собой, между программами и ОС и тд; эти регистры справа на картинке)
Регистры управления памятью
Добавлены регистры поддержки работы в защищенном режиме
GDTR: (Global Descriptor Table Register) 6-байтный регистр, содержит 32-битный линейный адрес начала таблицы глобальных дескрипторов (GDT) и 16-битный размер (лимит, уменьшенный на 1);
IDTR: (Interrupt Descriptor Table Descriptor; то есть в защищенном режиме таблица векторов прерываний начинается с некоторого произвольного адреса) 6-байтный регистр, содержит 32-битный линейный адрес начала таблицы глобальных дескрипторов обработчиков прерываний (IDT) и 16-битный размер (лимит, уменьшенный на 1);
LDTR: (Local Descriptor Table Register) 10-байтный регистр, содержит 16-битный селектор для GDT и весь 8-байтный дескриптор из GDT, описывающий текущую таблицу локальных дескрипторов;
TR: (Task Register) 10-байтный регистр, содержит 16-битный селектор для GDT и весь 8-байтный дескриптор из GDT, описывающий TSS текущей задачи.

26. Страничная модель памяти. Виртуальная память
Модели памяти
* Плоская - код и данные используют одно и то же пространство
* Сегментная - сложение сегмента и смещения (используется в реальном режиме; знакома нам)
* Страничная - виртуальные адреса отображаются на физическое постранично * **виртуальная память** - метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (файл, или раздел подкачки); * основной режим для большинства современных ОС; * в x86 минимальный размер страницы - 4096 байт; * основывается на таблице страниц - структуре данных, используемой системой виртуальной памяти в операционной системе компьютера для хранения сопоставления между виртуальным адресом и физическим адресом. Виртуальные адреса используются выполняющимся процессом (программа работает информация только с виртуальных адресах), в то время как физические адреса используются аппаратным обеспечением. Таблица страниц является ключевым компонентом преобразования виртуальных адресов, который необходим для доступа к данным в памяти.
Преимущества страничной модели:
* Программы полностью изолированы друг от друга
* В память можно загрузить больше программ, чем памяти доступно (долго не использующиеся данные загружаются на диск и освобождают место)
страничное преобразование
Линейный адрес * биты 31-22 - номер таблицы страниц в каталоге;
- биты 21-12 - номер страницы в выбранной таблице;
* биты 11-0 - смещение от физического адреса начала страницы в памяти.
Каждое обращение к памяти требует двух дополнительных обращений (проблема, долго); Необходим специальный кеш страниц - TLB (Translation Lookaside Buffer) - кеш, в котором хранятся последние использованные страницы (проблема, дорого);
Каталог таблиц/таблица страниц:
* биты 31-12 - биты 31-12 физического адреса таблицы страниц либо самой страницы;
* младшие биты - атрибуты управления страницей (если это таблица страниц, то элементы - страницы программы, если это каталог таблиц, то данные - таблицы страниц отдельных программ).



27. Математический сопроцессор. Типы данных
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).
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:
0 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.
-

28. Математический сопроцессор. Регистры
Предусмотрено 8 80-разрядных регистров данных (**ST0-ST7).** При этом, к ним напрямую адресовать нельзя. Адресация происходит с помощью стека, который реализован внутри этого сопроцессора. 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, **Стек закольцован и** имеет 8 элементов.
SR - регистр состояний, содержит слово состояния FPU. Сигнализирует о различных ошибках, переполнениях. Отдельные биты описывают и состояния регистров 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 **SR, CR, TW - 16-разрядные.**

29. Математический сопроцессор. Классификация команд.
Команды пересылки данных:
* команды взаимодействия со стеком (загрузка - выгрузка вещественного числа, целого, BCD (упакованного); смена мест регистров)
Команды пересылки данных FPU
● **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
Базовая арифметика FPU
.DATA
num1 REAL4 3.14 ; Первое число с плавающей запятой
num2 REAL4 2.71 ; Второе число с плавающей запятой
result REAL4 ? ; Результат сложения
.CODE
FADD num1, num2 ; Сложение чисел с плавающей запятой
FSTP result ; Сохранение результата в памяти
● **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) заменяется на мантиссу и экспоненту, мантисса дописывается на вершину стека ● **FSQRT** - вычисляет квадратный корень ST(0) **Команды сравнения FPU**
сравнивают 2 числа и выставляют флаги (основные и в регистре SR), некоторые команды выталкивают после сравнения числа из стека. Также можно сравнить только значащих разрядов (мантисс, то есть без порядков), сравнение только целых частей, аналог TEST (сравнение с нулем), а также команда выставления флагов в зависимости от типа числа.
● **FCOM, FCOMP, FCOMPP** - сравнить и вытолкнуть из стека
● **FUCOM, FUCOMP, FUCOMPP** - сравнить без учёта порядков и вытолкнуть
● **FICOM, FICOMP, FICOMP** - сравнить целые
● **FTST** - сравнивает с нулём
● **FXAM -** выставляет флаги в соответствии с типом числа
Трансцендентные операции сопроцессора:
sin (fsin), cos(fsin) - принимают значение в радианах в некотором диапазоне. Операндом обычно выступает значение в *радианах*, при этом не во всем диапазоне (нельзя давать слишком большие углы).
Константы FPU: Загрузка констант на вершину стека 1 FLD1 **FLDZ -** загрузить 0 **FLDPI** - загрузить PI **FLDL2E** - загрузить log2(e) **FLDL2T** - загрузить log2(10) **FLDLG2** - загрузить lg(2) **FLDLN2** - загрузить ln(2)
Команды управления: Предусмотрены команды прямого изменения указателя вершины стека, сохранения/изменения состояние сопроцессора. По-хорошему, FINIT должна вызываться первой, при желании поработать с сопроцессором (она очистит все регистры от данных, пометив их пустыми, регистры CR и SR - значениями по умолчанию). FSAVE и FRSTOR сохраняет 94 или 108 байт в зависимости о разрядности машины.
FINCSTP, FDECSTP - увеличить/уменьшить указатель вершины стека
FFREE - пометить регистр как пустой
FINIT, FNINIT - инициализировать дефолтными значениями
FCLEX, FNCLEX - очистить регистры CR и SR и флаги исключений .

30. Расширения процессора. MMX. Регистры, поддержива типы данных
Расширение, которое было встроено для увеличения эффективности обработки больших потоков данных (простые операции над массивами однотипных данных (звук, изображения, видеопоток))
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.
Типы данных MMX:
- учетверенное слово (64 бита);
- упакованные двойные слова (2 32-разряд слова);
- упакованные слова (4 слова в одном 64бит регис);
- упакованные байты (8 байт в одном 64бит регист).

31. Расширения процессора. MMX. Классификация команд
Насыщение - замена переполнения/антипереполнения превращением в максимальное/минимальное значение (Светлый цвет + светлый цвет максимально белый (но не будет переполнения и темного цвета))
Команды пересылки данных MMX
● MOVD, MOVQ - пересылка двойных/учетверённых слов
● PACKSSWB, PACKSSDW - упаковка со знаковым насыщением слов в байты/двойных слов в слова. Приёмник -> младшая половина приёмника, источник -> старшая половина приемника packssdw mm0, mm1; Упаковать 32-битные целые числа из mm0 и mm1 в 16-битные целые числа в mm0
● PACKUSWB - упаковка слов в байты с беззнаковым насыщением
● PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ - распаковка и объединение старших элементов источника и приёмника через 1 **punpckhbw mm0, mm1** ; Распаковать старшие 8-битные элементы из mm0 и mm1 через 1 элемент и сохранить результат в mm0
Арифметические операции MMX
● PADDB, PADDW, PADDD - поэлементное сложение, перенос игнорируется
● PADDSB, PADDSW - сложение с насыщением
● PADDUSB, PADDUSW - беззнаковое сложение с насыщением
● PSUBB, PSUBW, PDUBD - вычитание, заём игнорируется
● PSUBSB, PSUBSW - вычитание с насыщением
● PSUBUSB, PSUBUSW - беззнаковое вычитание с насыщением
● PMILHW, PMULLW - старшее/младшее умножение (сохраняет старшую или младшую части результата в приёмник)
● PMADDWD - умножение и сложение. Перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших ```psubb mm0, mm1 ; Вычитание байтовых элементов mm1 из соответствующих элементов mm0 и сохранение результата в mm0
Команды сравнения MMX
● PCMPEQB, PCMPEQW, PCMPEQD - проверка на равенство. Если пара равна - соответствующий элемент приёмника заполняется единицами, иначе - нулями ```pcmpeqb mm0, mm1
● PCMPGTB, PCMPGTW, PCMPGTD - сравнение. Если элемент приёмника больше, то заполняется единицами, иначе - нулями
Логические операции MMX
● PAND - логическое И
● PANDN - логическое НЕ-И (штрих Шеффера) (источник*НЕ(приёмник)) pand mm0, mm1 ;
● POR - логическое ИЛИ ● PXOR - исключающее ИЛИ
Сдвиговые операции MMX
● PSLLW, PSLLD, PSLLQ - логический влево
● PSRLW, PSRLD, PSRLQ - логический вправо
● PSRAW, PSRAD - арифметический вправо(со знаком)

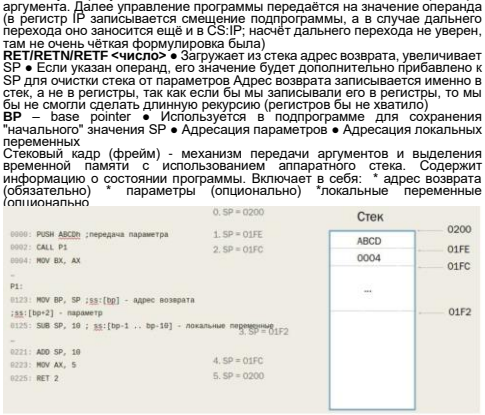
32. расшир процес. SSE. Регистры типы данных
После MMX появилось расширение SSE решило проблему параллельной работы с FPU. ● Single Instruction, Multiple Data: одна инструкция — множество данных ● 8 128-разрядных регистровxmm0 — xmm7
● Свой регистр флагов ● Основной тип - вещественные одинарной точности (32 бита) ● Целочисленные команды работают с регистрами MMX
типы данных
1. Упакованные однобайтовые целые числа (16)
2. Упакованные слова (8) 3.Упакованные двойные слова (4)
4. Одинарная точность с плавающей запятой (4)
5.Двойная точность с плавающей запятой (2)

33. Расширения процессора. SSE. Классификация команд
○ Пересылки MOVAPS / MOVUPS (без выравнивания)
.DATA num1 REAL4 1.5, 2.5, 3.5, 4.5
.CODE MOVAPS XMM0, num1
○ Арифметические ADDPS 4 числа float / SUBPD 2 float
○ Сравнения CMPPS 4 float
○ Преобразования типов CVTPS2PI 4 числа с плавающей запятой одинарной точности в 4 целых числа.
○ Логические ANDPS XORPD
○ Целочисленные PADDW: Выполняет сложение 8 слов
○ Упаковки PACKSSDW: Упаковывает 4 двойных слова (32-битных целых числа) в 8 слов (16-битных целых чисел).
○ Управления состоянием EMMS: Восстанавливает состояние регистров FPU после использования инструкций SSE.
○ Управления кэшированием PREFETCHNTA: Предварительно загружает данные в кэш без ограничений на тип доступа

34. Расшир процес. AVX. Регистры, поддерж типы данных.
● регистры увеличены со 128 (XMM) до 256 (YMM0-YMM15) бит: YMM0-YMM15: **16 регистров** общего назначения размером 256 бит (32 байта), используемых для хранения векторных данных. Каждый регистр YMM может содержать 8 значений одинарной точности (float) или 4 значения двойной точности (double).
XMM0-XMM15: **16 регистро**в общего назначения размером 128 бит (16 байт), также используемых для хранения векторных данных. Каждый регистр XMM может содержать 4 значения одинарной точности (float) или 2 значения двойной точности (double) ● SSE-инструкции используют младшую половину YMM-регистров, не меняя старшую часть; ● "неразрушающие" (трёхоперандные) инструкции: c = a + b вместо a = a + b;
● AVX3 (2013 г.) - 512-битное расширение
типы данных Можно использовать в регистрах XMM и YMM.
Одинарная точность : float: 32-битные числа с плавающей **Двойная точность** double: 64-битные числа с плавающей **Целочисленные типы данных:** 8-битные целые числа (**byte**): signed char, unsigned char. 16-битные целые числа (**word**): short, unsigned short. 32-битные целые числа (**dword**): int, unsigned int.. 64-битные целые числа (**qword**): long long, unsigned long long. **Логические типы данных**: 32-битные логические значения (dword): 64-битные логические значения (qword):

18. Виды ассемблеров. Intel-синтаксис, AT&T-синтаксис.
Транслятор ассемблера - программа, которая переводит текст программы на языке ассемблера в машинный код.
Виды трансляторов ассемблера MASM TASM NASM FASM YASM
MOV DX, OFFSET MESSAGE **Особенности Intel-синтаксиса**
В команде приёмник находится слева от источника (например: mov eax,ebx, где mov-это команда, а eax и ebx операнды этой команды, один из них eax-приемник, ebx-источник). * Название регистров зарезервировано (нельзя использовать метки с именами eax, ebx и т. д.)
AT&T-синтаксис
Синтаксис стандартного ассемблера для UNIX - as
1. Имена регистров предваряются префиксом %. *(mov %eax,%ebx)*
2. Обратный порядок операндов: вначале источник, затем приёмник.
3. Размер операнда задается суффиксом, замыкающим инструкцию.
например: movl %eax,%ebx, где l-это суффикс, обозначающий размер операнда в 32 бита)
4. Числовые константы записываются в Си-соглашении.
5. Для получения смещения метки используется префикс $
mov $6, %eax # Загрузка числа 6 в регистр eax
addl $5, %eax // Сложение числа 5 с содержимым регистра

19 - 20. Подпрограммы, Объявление, вызов возврат
CALL <операнд> ● Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по адресу IP либо CS:IP, в зависимости от размера аргумента) ● Передаёт управление на значение аргумента. Она похожа на **PUSH** (но немного сложнее): сохраняет адрес следующей за ней команды в стеке (той, которая физически за ней стоит в памяти, а не та, которая вызывается при переходе в подпрограмму), уменьшает SP (в случае ближнего перехода на 2 байта, в случае дальнего - на 4 байта) и записывает по этому адресу значение IP либо CS:IP, в зависимости от размера аргумента. Далее управление программы передаётся на значение операнда (в регистр IP записывается смещение подпрограммы, а в случае дальнего перехода оно заносится ещё и в CS:IP); насчёт дальнего перехода не уверен, там не очень чёткая формулировка была)
RET/RETN/RETF <число> ● Загружает из стека адрес возврата, увеличивает SP ● Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров Адрес возврата записывается именно в стек, а не в регистры, так как если бы мы записывали его в регистры, то мы бы не смогли сделать длинную рекурсию (регистров бы не хватило)
BP — base pointer ● Используется в подпрограмме для сохранения "начального" значения SP ● Адресация параметров ● Адресация локальных переменных
Стековый кадр (фрейм) - механизм передачи аргументов и выделения временной памяти с использованием аппаратного стека. Содержит информацию о состоянии программы. Включает в себя: * адрес возврата (обязательно) * параметры (опционально) *локальные переменные (опционально)



21. Макроопределения
Макроопределение (макрос) - именованный участок программы, который ассемблируется каждый раз, когда его имя встречается в тексте программы
Макроопределение - специальным образом оформленная последовательность предложений языка ассемблера, под управлением которой ассемблер (точнее, его часть, называемая макрогенератором или препроцессором) порождает макрорасширения макрокоманд
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.
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
Директива присваивания = служит для создания целочисленной макропеременной или изменения её 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
Пример:
X EQU [EBP+8]
MOV ESI,X
Макроопределения. Макрооперации. % - вычисление выражение перед представлением числа в символьной форме
MP_REC MACRO P
MOV AX,P
IF P
MP_REC %(P-1) ;;перед записью в МРасш-ние вычислить P-1
ENDIF
ENDM
<> - подстановка текста без изменений (полезно, когда есть вероятность пересечения имени какого-либо макроса с простым (или не очень) текстом, которым мы хотим вставить) & - склейка текста (A&B ==> AB, если параметры - макропараметры, то они склеятся) **!** - считать следующий символ текстом, а не знаком операции (AI&B ==> A&B) **;;** - исключение строки из макроса (После препроцессора эта строчка исчезнет (если одна ";", то комментарий остается"
IFB <par> - истинно, если параметр не определён (то есть фактический параметр рег 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
22. Стек. Аппаратная поддержка вызова подпрограмм.
Стек работает по правилу **LIFO / FILO** (последним пришёл, первым вышел)
Сегмент стека — область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний. Используется для временного хранения переменных, передачи параметров для 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.
Команды: PUSH <источник> - поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
POP <приёмник> - считать данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
PUSHA - поместить в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
POPA - загрузить регистры из стека (SP игнорируется)
PUSHF - поместить в стек содержимое регистра флагов
POPF - загрузить регистр флагов из стека
Основные компоненты **аппаратной поддержки вызова** подпрограмм включают: **Стековый указатель** (Stack Pointer, SP): Регистр, который содержит адрес вершины стека. Он указывает на текущую свободную ячейку на стеке. **Регистр базы** стека (Stack Base Pointer, BP): Регистр, который используется для доступа к параметрам и локальным переменным внутри подпрограммы. Он обычно указывает на фиксированный адрес в области стека, называемой фреймом или стековым фреймом
```
My_Subroutine:
    push   bp          ; Save old BP value
    mov    bp, sp      ; Set BP to current SP
    ; In 16-bit mode:
    mov    bx, [bp + 4]    ; arg1
    mov    cx, [bp + 6]    ; arg2
    mov    dx, [bp + 8]    ; arg3
    mov    sp, bp      ; Restore SP
    pop    bp          ; Restore BP
    ret                ; Return to caller
```

23. Соглашения о вызовах. Понятие, основные виды соглашений
Описания технических особенностей вызова подпрограмм, определяющие:
● способы передачи параметров подпрограммам;
● способы передачи управления подпрограмм;
●способы передачи результатов выполнения из подпрограмм в точку вызова; ● способы возврата управления из подпрограмм в точку вызова
Распространённые соглашения
● cdecl ● pascal ● stdcall (WinAPI) ● fastcall ● safecall ● thiscall
Соглашение о вызове cdecl: Это распространенное соглашение в стековом стиле, которое используется в С-подобных языках. Аргументы передаются через стек в обратном порядке (справа налево). Вызывающая сторона очищает аргументы со стека. Возврат значения происходит через регистр EAX

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 усл и безусловной передачи управления.
JMP <операнд> - оператор безусловного перехода. Передает управление в другую точку программы, не сохраняя какой-либо информации для возврата. Операнд - непосредственный адрес, регистр или переменная.
вид переода дистанция перехода
short -128..127 байт (изменяем IP-смещение без CS)
near в том же сегменте (без изменения CS)
far 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).
Команды условных переходов
Переход типа short или near. Обычно используются в паре с CMP "Выше" и "ниже" - при сравнении беззнаковых чисел "Больше" и "меньше" - при сравнении чисел со знаком

Команда	Описание	Состояние
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 = 1
JNE/JNZ	Не равно/не ноль	ZF = 0
JP/JPE	Есть чётность/чётное	PF = 1
JNP/JPO	Нет чётности/нечётное	PF = 0
JCXZ	CX = 0	

Команда	Описание	Состояние флагов для выполнения перехода	Знако
JB	Если ниже	CF = 1	Нет
JNAE	Если не выше и не равно	CF = 1	Нет
JC	Если перенос	CF = 1	Нет
JNB	Если не ниже	CF = 0	Нет
JAE	Если выше или равно	CF = 0	Нет
JNC	Если нет переноса	CF = 0	Нет
JBE	Есть ниже или равно	CF = 1 или ZF = 1	Нет
JNA	Если не выше	CF = 1 или ZF = 1	Нет
JA	Если выше	CF = 0 и ZF = 0	Нет
JNBE	Если не ниже и не равно	CF = 0 и ZF = 0	Нет
JL	Если меньше	SF <> OF	Да
JNGE	Если не больше и не равно	SF <> OF	Да
JGE	Если больше или равно	SF = OF	Да
JNL	Если не меньше	SF = OF	Да
JLE	Если меньше или равно	ZF = 1 или SF <> OF	Да
JNG	Если не больше	ZF = 1 или SF <> OF	Да
JG	Если больше	ZF = 0 и SF = OF	Да
JNLE	Если не меньше и не равно	ZF = 0 и SF = OF	Да

11. Арифметические команды. **ADD <приёмник>, <источник>** Сложение приемника и источника. Сумма - в приемник, источник не изменяется. **Приемник** - переменная (область памяти), РОН **Источник** - тоже самое что приемник или непосредственный операнд (например, число)
SUB <приёмник>, <источник> Вычитание, всё точно так же как и в ADD. Отрицательные числа можно отслеживать с помощью флага SF.**MUL <источник> Умножение без знака. Источник** - область памяти, РОН. Умножают источник и AL|AX, в зависимости от размера (разрядности) источника (если 1 байт, то AL, а если 2 байта, то на AX). Результат помещается в AX (если 1 байт умножался на 1 байт; то есть предполагается, что результат может быть в 2 раза длиннее аргумента) либо DX:AX (если 2 байта умножались на 2 байта; предполагается, что результат получится 4-байтовым, и его старшие 2 байта помещаются в DX, младшие 2 - в AX **DIV <источник>** - целочисленное беззнаковое деление. Делится AL/AX на источник. Результат помещается в AL/AX, остаток - в AH/DX. Выбор регистра опять зависит от разрядности источника Ещё есть команды деления/умножения со знаком: **IDIV, IMUL DEC (-) <операнд>, INC (+)<операнд>** Обе команды работают быстрее ADD и SUB соответственно, потому что занимают 1 байт, а не 3. INC и DEC, в отличии от ADD и SUB, не затрагивают флаг CF. Все эти команды меняют регистр флагов (FLAGS), в зависимости от результата

12. Двоично-десятичная арифметика
Двоично-десятичные числа используются в аппаратуре, которая должна на дисплей выводить десятичные числа, не занимаясь доп вычислениями (переводом). 1 -0001 … 9 - 1001
Неупакованное двоично-десятичное число - байт от 00h до 09h. 1 байт - число 10011 0001 0001_{BCD}. = 311_{10}
Упакованное двично-десятичное число - байт от 00h до 99h (цифры A..F не задействуются). В упакованном формате используется один байт для представления двух десятичных цифр. 6.75 будет 0110.01110101.
При выполнении арифметических операций необходима коррекция: 19h + 1 = 1Ah => 20h
DAA, DAS - Коррекция упакованных двоично-десятичных чисел после сложения или вычитания.
```
mov AL, 71H    ; AL = 0x71h
add AL, 44H    ; AL = 0x71h + 0x44h = 0xB5h
daa            ; AL = 0x15h
  ; CF = 1 - перенос является частью результата 71 + 44 = 115
```
Команда AAA (ASCII Adjust After Addition) - позволяет преобразовать результат сложения BCD-чисел в ASCII формат. Для того чтобы преобразовать содержимое регистра AL к ASCII формату, необходимо после команды AAA выполнить команду OR AL, 0x30h. Команда AAA должна выполняться после команды ADD, которая помещает результат сложения в AL. Если будет перенос, он записывается в AH.
```
xor ah, ah
add al, '6'    ; al = 0x36h
add al, '8'    ; al = 0x36h + 0x38h = 0x6Eh
aaa            ; ax = 0x0104h
or al, 0x30h   ; al = 0x34h = '4'
```
Команда AAS (ASCII Adjust After Subtraction) - позволяет преобразовать результат вычитания BCD-чисел в ASCII формат. Для этого команда AAS должна выполняться после операнды SUB, которая помещает 1 байтный результат вычитания в AL. Если был заем, то 1 вычитается из AH.
```
; положительный результат
xor ah, ah
mov al, '9'    ; al = 0x39h
sub al, '3'    ; al = 0x39h - 0x33h = 0x06h
aas            ; ax = 0x0006h
or al, 0x30h   ; al = 0x36h = '6'
; отрицательный результат
xor ah, ah
mov al, '3'    ; al = 0x33h
sub al, '9'    ; al = 0x33h - 0x39h = 0xFAh
aas            ; AX = 0xFF04h
or al, 0x30h   ; al = 0x34h = '4'
```
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).Команда AAM распаковывает результат умножения, содержащийся в регистре AL, деля его на второй байт кода операции ib которая помещает двухбайтный результат в регистр AX. Частное от деления (наиболее значащая цифра результата) помещается в регистр AH, а остаток (наименее значащая цифра результата) — в регистр AL.
```
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
```
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.
```
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'
```
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.

35. Расширения процессора. AVX. Классификация команд
1. Арифметические команды: VADDPS (добавление скалярных пакетных с чисел с плавающей точкой), VMULPS (умножение скалярных пакетных с чисел с плавающей точкой)
2. Сравнительные команды: VCMPPS (сравнение упакованных чисел с плавающей точкой)
3. Логические команды: (VANDPS, VANDPD, VANDNPS, VANDNPD) (VORPS, VORPD, VXORPS, VXORPD)
4. Преобразовательные команды: (VCVTPS2PD, VCVTDQ2PS, VCVTSS2SD)
5.Упаковка и распаковка: Команды, которые переставляют элементы данных из одного вектора в другой или объединяют данные из нескольких источников.VPACKSSWB (упаковка знаковых упакованных слов в знаковые упакованные байты), VUNPCKLPS (распаковка нижних упакованных чисел с плавающей точкой
6. Команды перестановки и перемешивания: Эти команды изменяют порядок элементов данных, перемешивая или маскируя их в разных источниках и назначениях. Например, команды VSHUFPS (перемешивание упакованных чисел с плавающей точкой), VPERMILPS (перестановка упакованных чисел с плавающей точкой) и так далее.
7. Команды пересылки данных: (VMOVAPS, VMOVUPS, VMOVAPD, VMOVUPD) для загрузки и сохранения данных между регистрами и памятью.

36. Процессоры семейс x86-64. Регистры, режимы работы.
Регистры
- Целочисленные 64-битные регистры общего назначения: RAX,RBX,RCX,RDX,RSI,RDI,RBP,RSP
- Новые целочисленные 64-битные регистры общего назначения: R8… R15
- 64-битный указатель RIP (Instruction Pointer) - указатель на следующую инструкцию
- 64-битный регистр флагов RFLAGS
- Регистры масок, такие как K0-K7, используются в расширении AVX-512 для контроля активности элементов в векторных регистрах

Legacy Mode - режим совместимости с 32-разрядными процессорами №унаследованный№
Long Mode - №родной№ 64-разрядный режим с частичной поддержкой 32-разрядных программ (32разрядный код не должен пересекаться с 64разрядным). Рудименты V86 и сегментной модели памяти упразднены (DOS программы нельзя теперь запустить, только на 32разрядной системе можно запустить).

37. Расшир проце. AES. Назначение, классификац команд
Расширение AES (Intel Advanced Encryption Standard New Instructions; AES-NI, 2008) Реализует алгоритмы шифрования AES (Advanced Encryption Standard) и Galois/Counter Mode (GCM) для ускорения шифрования и расшифрования данных.
Реализует алгоритмы хэширования SHA-1 и SHA-256 для ускорения вычисления хэш-функций.
Классификация команд
- раунда шифрования AESENC (AES Encrypt)
- раунда расшифрования т AESDEC (AES Decrypt),
- раунда генерации ключа AESKEYGENASSIST,

38. Архитектура RISC. Семейство процессоров ARM.
Версии архитектуры, профайлы.
представляет собой подход к проектированию процессоров, в котором используется набор простых и однородных команд с фиксированной длиной. Это позволяет упростить аппаратную реализацию процессора и повысить его производительность.
Ранние архитектуры процессоров (комплексные, CISC (Complex Instruction Set Computer)):
- большое количество команд
- разные способы адресации для упрощения написания программ на ассемблере
- поддержка конструкций языков высокого уровня

Недостатки: на практике многие возможности CISC используются компиляторами ЯВУ ограниченно, а их поддержка затратна.
Архитектуры RISC (Reduced Instruction Set Computer):
- сведение набора команд к простым типовым
- большое количество регистров (возможно за счет общего упрощения архитектуры)
- стандартизация форматов команд, упрощение конвейеризации

Семейство процессоров ARM
Процессоры ARM (Advanced RISC Machine) занимают 90% рынка мобильных устройств.
Версии архитектуры
ARMv1 - 32-битная архитектура (26-битное адресное пространство), 1985 год
ARMv7 - 32-битная архитектура, 2004 год расширенные возможности обработки плавающей запятой и поддержку технологии NEON для обработки сигналов и мультимедиа.
ARMv8 - 64-битная архитектура, 2011 год
ARMv9 - 64-битная архитектура, 2021 год. Перспективная архитектура с поддержкой векторных инструкций SVE2
Профайлы - наборы расширений архитектуры, определяющие набор команд и режимы работы процессора.
Classic Этот профиль обеспечивает общие функциональные возможности, архитектурные режимы и базовые наборы инструкций. **Real-time** Он включает набор команд и режимов, которые обеспечивают низкую задержку выполнения и возможность работы с жесткими сроками.
Microcontroller Он обеспечивает набор команд и режимов, которые учитывают ограниченные ресурсы встраиваемых систем, такие как ограниченное объем памяти и низкое энергопотребление. **Application** file предназначен для общего назначения и предоставляет широкий набор команд и режимов, оптимизированных для различных приложений.

39. Процессо ARM. Наборы команд. Основные регистры.
Наборы команд
- ARM - 32-битный набор команд
- Thumb - 16-битный набор команд
- Thumb2 - 16/32-битный набор команд
- A64 - 32-битный набор

Архитектура ARMv8 включает в себя:
● 31 64-битный регистр R0..R30. Доступ осуществляется по алиасам X0..X30 и по алиасам W0..W30 для младших 32-битных половин. При записи значений в регистры W0..W30 старшие половины регистров зануляются, в отличие от x86;
● PC (program counter) - программный счётчик, аналог IP в x86;
● LR (алиас для X30) - ссылочный регистр. Хранит в себе последний адрес возврата;
● SP - указатель стека;
● V0..V31 - 128-битные SIMD-регистры расширения NEON (аналог SSE в x86).Доступ осуществляется по алиасам Q0..Q31 для всех 128 бит и по другим алиасам для младших частей.
CPSR (Current Status Program Register) — это 32-битный регистр, используемый в архитектуре ARM для хранения информации о состоянии процессора и текущем режиме выполнения. CPSR содержит флаги состояния процессора, такие как флаги условий, флаг переноса, флаги прерывания и другие.**Флаги условий** (Condition Flags) — это флаги, которые устанавливаются в зависимости от результата выполнения инструкций. **N (Negative)** — устанавливается, если результат операции отрицательный. **Z (Zero)** — устанавливается, если результат операции равен нулю. **C (Carry)** — устанавливается, если результат операции превышает диапазон. **V (Overflow)** — устанавливается, если результат операции приводит к переполнению **CPSR** также содержит информацию о текущем режиме выполнения, таком как режим пользователя (User mode), режим привилегированного доступа (Privileged mode), режим системного (System mode) и другие.

40. Процессоры ARM. Основные команды
Основные команды
Команды пересылки данных:
- MOV - пересылка данных между регистрами
- LDR - загрузка данных из памяти в регистр
- STR r3, [r4]- сохране данных из регистра в память
-

Команды арифметических операций:
- ADD - сложение ADD r1, r2, r3
- SUB - вычитание
- MUL - умножение

Команды деления отсутствуют. Замена для деления на константу - умножение на заранее вычисленную степень 2, затем сдвиг вправо.
Команды логических операций:
- AND - логическое И AND r1, r2, r3 ;
- ORR - логическое ИЛИ
- XOR - логическое исключающее ИЛИ
- LSLS - сдвиг влево
- LSRS - сдвиг вправо
- ASR - арифметический сдвиг вправо
- ROR - циклический сдвиг вправо
- RRX - циклический сдвиг вправо с расширением знака

Команды сравнения:
- CMP - сравнениет CMP r1, #0 сравнение с нулем

Команды ветвления:
- B - безусловный переход B label
- BL (Branch with Link) - переход с сохранением адреса возврата в LR
- BLX (Branch with Link and Exchange) - переход с переключением системы команд.

Допускаются команды push lr и pop pc.
Вызов программного прерывания:
SWI - вызов программного прерывания
SQI imned_8(0..255) - вызов программного прерывания с указанием номера прерывания Переводит процессор в Supervision Mode, CPSR сохраняется в Supervisor Mode SPSR, управление передается обработчику прерывания по вектору.