#### Лекция 1.«Системы счисления»

**Системы счисления:**

  2-я использует цифры: 0,1  - проще для ЭВМ

10-я использует цифры: 0,1,2,3,4,5,6,7,8,9  - проще для человека

16-я использует цифры: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F  - для удобства общения ЭВМ и человека

**2-я  10-я 16-я**  
0000    0      0  
0001    1      1  
0010    2      2  
0011    3      3  
0100    4      4  
0101    5      5  
0110    6      6  
0111    7      7  
1000    8      8  
1001    9      9  
1010  10      A  
1011  11      B  
1100  12      C  
1101  13      D  
1110  14      E  
1111  15      F

**Перевод чисел в разные системы счисления:**

**1) ???10**→**???n**   - деление столбиком (делим, затем выписываем остатки в обратном порядке)

 12510 → ???7 = 2367      12510 → ???16 = 7D16  (13 – смотрим таблицу – цифра D)

Деление происходит до тех пор, пока не получится результат, меньший n - основание системы счисления. Если остаток получился  >10  – смотрим нашу таблицу.

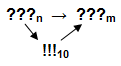
**2) ???n**→**???10**

 07 06 15 04 03 12 01 102 → ??10    ─ нумеруем разряды по старшинству (с 0-го по 7-й)

 0\*27 + 0\*26 + 1\*25 + 0\*24 + 0\*23 + 1\*22 + 0\*21 + 1\*20 = 25 + 22 + 21 = 32 + 4 + 1 = 3710

 71 D016 → ??10

 7\*161 + D\*160 = 7\*16 + 13\*1 = 112 + 13 = 12510   ─ верно, проверьте на калькуляторе

**3)** 

Из произвольной в произвольную - за 2 шага, через промежуточную 10-ю!

**4) ????2 ↔ ?16** - быстрый перевод (используется, что 16 = 24)

Каждому шестнадцатеричному разряду строго соответствует четыре двоичных (и наоборот)!

 1011002  →  \_ \_ 10 11002  →  0010 11002  →  2C16   ─ разбивка на четверки, дополняем старшие нули и замена по таблице

6F516 → 0110 1111 01012  → 110 1111 01012   ─ замена по таблице (1 цифра = 1 четверка бит) и убираем старшие нули

**Простейшие арифметические операции в 2-й системе:**

**1) Сложение: 510 + 710 = 1210**

510 →    000001012  
710 → + 00000111 2  
               000011002  
                          <<<    ─  переносы в старший разряд

**2) Умножение:**

При сдвиге двоичного числа на 1 разряд влево, оно удваивается.

При сдвиге двоичного числа на n разрядов влево, оно умножается на 2n.

Вправо - происходит целочисленное деление (с отбрасыванием остатка).

 710 \* 510 = 1112 \* 1012 = 1112 \* (22 + 20) = 111002 + 1112 = 001000112 = 25 + 21 + 20 = 32 + 2 + 1 = 3510

**3) Вычитание: 710 - 510 = 710 + (-5дополнительный код) = 210**

 510 →    000001012  →|  
               111110102  ←|    ─ инверсия  
           +                 12  
              ----------------  
-5доп      111110112  
710 → + 000001112  
          -------------------  
            **1** 000000102 = 210

Возникшее переполнение (9-й разряд) ─ отбрасывается!

510 - 710 = 510 + (-7дополнительный код) = -210

710 →    000001112  →|  
              111110002  ←|    ─ инверсия  
            +                12  
            -----------------  
-7доп      111110012  
510 → + 000001012  
            -----------------  
-2доп      111111102  →|  
               000000012 ←|    ─ инверсия  
            +                12  
             -----------------  
               **0**00000102   ─ верно, совпадает

 В машинном представлении чисел старший разряд (№ 7) - знаковый, показывает знак числа ("0" - плюс, "1" - минус). Если результат операции - отрицательный, значит - он записан в дополнительном коде (для проверки надо преобразовать обратно).

#### Лекция 2. «Введение в низкоуровневое программирование»

Язык **Assembler** – представляет собой набор элементарных команд, выполняемых процессором напрямую. С одной стороны, это гарантирует максимальную скорость работы программ, с другой – для реализации нетривиальных задач требуются значительные усилия. Кроме того, необходимо учитывать, что в разных процессорах используются разные наборы команд (которые и кодируются по-разному). Присутствует проблема переносимости ПО.

**Рассмотрим краткую историю МП-техники на примере семейства Intel:**

**8086   -** Одним из первых был процессор 8086. Он был 16 битным, мог работать с 1 Мб ОЗУ.

**80186 -** Промежуточный вариант, который быстро был вытеснен следующим представителем.

**80286 -**Позволял работать в 2 режимах:

\* реальный режим (загружается по умолчанию)

\* защищенный режим (доступна вся память – снимаются все ограничения, пользователь сам отвечает за любые свои действия)

**80386 -**Данная серия процессоров была полностью 32-битовой, поддерживала 3 режима работы:

\* **реальный режим** (то, что мы будем изучать)

\* защищенный режим 2 (по сравнению с предыдущ. добавлена таблица новых команд)

\* виртуальный режим 386 процессора (впервые появляется мультизадачность)

Отметим, что далее повышаются характеристики процессоров: быстродействие, разрядность; добавляются новые функции. Мы остановимся именно на реальном режиме 386 процессора, поскольку он корректно воспринимается всеми последующими представителями данной серии.

**Адресация памяти:**

Вся память в ЭВМ представляет собой аналог одномерного массива – длинная лента, состоящая из элементарных/битовых ячеек (значения 0 или 1). 8 бит называют байтом, а 2 байта – машинным словом. Поскольку емкость памяти современных запоминающих устройств огромна, память поделена для удобства на небольшие порции – сегменты. Каждый сегмент является независимым от других, и может быть использован для загрузки программы при ее выполнении или для хранения каких-либо данных. В общем случае, если информация не умещается в одном сегменте, ей могут быть выделены другие. Но нам и одного сегмента хватит с избытком!

1 байт – самая мелкая ячейка памяти, которую можно адресовать с помощью стандартной системы адресации. При этом, адрес состоит из 2 частей: номер сегмента и смещение (т.е. на сколько надо внутри отступить от начала этого сегмента). Формат адреса записывается в виде восьми 16-ричных цифр, и хорошо знаком вам по сообщениям «Критическая ошибка Windows».

**Пример:** Адрес = Сегмент + Смещение = SEG : SME = 0012 : 4F7B

 На самом деле, ЭВМ рассчитывает абсолютный 20-разрядный адрес любой ячейки по формуле  A = 24 \* SEG + SME, но нас это пока не будет интересовать.

**Машинное представление информации в ячейке памяти**

 Машинное слово:  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]    [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]   - старшие байты идут правее младших

                                    7                             0    15                            8

                                         <-----------------                <------------------   старшинство разрядов в байтах

                                                  --------------------------------------->   старшинство байт в машинном слове

Кроме того, существуют специальные ячейки памяти, соединенные с процессором напрямую – **регистры**. У них нет адреса (находятся вне ОЗУ), обращаться к ним можно только по именам. Все операции с ними производятся в тысячи раз быстрее, чем с обычными ячейками.

В ЭВМ 8086 - 8186 было 14 регистров. Они делятся на служебные и общего назначения Служебные регистры: сегментные (указывают на начало сегментов в памяти) и указатели (указывают текущее положение внутри сегментов – смещение от начала). См. рисунок:

                       CS                      DS                              ES                                             SS

                         |------  IP             |------------  SI           |------- DI                                  |--- SP + BP

                         |         |               |                 |            |        |                                     |     |

                         V         V               V                 V            V       V                                     V     V

  ... память ... **][** сегмент кода **][** сегмент данных **][** экстракод – доп. сегмент **][** с т е к **][** ... память …

**Сегментные регистры:**

**CS** - сегмент кода (область памяти, куда загружен код программы)

**DS** - сегмент данных (область памяти, куда программа обращается за данными)

**ES** - экстракодовый сегмент (дополнительный сегмент, если мало двух предыдущих)

**SS** - сегмент стека (временное хранение промежуточных результатов – освобождаем регистры)

**Стек** – особая логическая структура, осуществляющая хранение данных по принципу **”последним пришел – первым ушел”** (аналогия – стакан).

В каждый момент времени доступен только один – верхний элемент. При добавлении нового все остальные как бы сдвигаются, и он становится верхним. При извлечении верхнего элемента – обратная картина: все сдвигаются обратно, верхним становится бывший второй элемент.

При работе со стеком могут произойти 2 ошибки: 1 – попытка извлечь элемент из пустого стека, 2 – попытка добавить элемент в полный стек (переполнение). Отслеживание таких ситуаций - задача пользователя. **Емкость стека – 64 Кб, размер хранимых элементов – 2 байта (16 бит).**

**Регистры - указатели:**

**IP** - (в паре с **CS**) указывает на ту инструкцию, которая должна выполниться следующей, после ее выполнения автоматически сместится на длину инструкции (изменяется с шагом +n)

**SP** - указатель стека, (в паре с **SS**) указывает на верхний элемент стека (изменяется с шагом ±2)

**BP**- указатель базы, также используется при работе со стеком

**DI**  - Destination Index register - (в паре с **ES**) индекс приемника – использ. в строковых операциях

**SI**  - Source Index register - (в паре с **DS**) индекс источника - используется в строковых операциях

**Регистры общего назначения** (доступны для решения задач пользователя):

**AX** (Accumulator register - аккумулятор) - обычно для хранения каких-либо промежуточных данных (используется по умолчанию при вызове прерываний и команд)

**BX** (Base register - база) - иногда для хранения базового адреса некоторого объекта в памяти

**CX** (Count register - счетчик) - в командах, производящих некоторые повторяющиеся действия (используется по умолчанию оператором цикла как параметр)

**DX** (Data register - регистр данных) - как и **AX**, хранит некоторые промежуточные данные (используется по умолчанию при вызове прерываний и команд)

В общем случае, эти регистры могут быть задействованы пользователем и не по назначению! Кроме того, для удобства, каждый их них разделен пополам:  старшие 8  +  младшие  8  разрядов.

Соответственно, их имена: **AH**, **AL**; **BH**, **BL**; **CH**, **CL**; **DH**, **DL** (H – high - старшие, L – low - младшие).

В командах языка можно обращаться как к отдельным половинкам, так и целым регистрам!

**Флаговый регистр FLAGS**(задействовано 9 бит – каждый отвечает за свое событие)**:**

**Флаг** (комп.) – ячейка памяти логического типа (1 бит), принимающая 1 из 2 значений: 0 или 1.

**OF** (Over Flow - флаг переполнения): равен 1, если в результате выполнения операции произошло переполнение в знаковый разряд. ПРИМЕР: 7FFF+1 = 8000 (было >0, стало <0).

**DF** (Direction Flag - флаг направления):  пока не нужен

**IF**  (Interrupt Flag - флаг разрешения прерывания):  пока не нужен

**TF** (Trace Flag - флаг трассировки):  пока не нужен

**SF** (Sign Flag - флаг знака): равен 1, если результат операции < 0 (отриц.). ПРИМЕР:  01 - 02 = FF

**ZF** (Zero Flag - флаг нуля): равен 1, если результат операции равен нулю.  ПРИМЕР:  1A - 1A = 00

**AF** (Additional carry Flag - флаг дополн. переноса):  пока не нужен (в двоично-десятичном коде)

**PF** (Parity Flag - флаг четности):  пока не нужен

**CF** (Carry Flag - флагом переноса):  равен 1, если в результате выполнения операции результат вышел за пределы разрядной сетки (17-й разряд). ПРИМЕР:  FFFF + 1 = 1 0000

#### Лекция 3. «Команды языка Assembler»

Поскольку исполняемый код программы является последовательностью целых чисел, мало понятных для пользователя, мы будем использовать специально разработанную программу-дебаггер, позволяющую писать команды в текстовом виде (**мнемокод**), и затем преобразующую их в числовое представление (и наоборот).

Все команды Assembler-а делятся на целочисленные и нецелочисленные (с плавающей точкой). Целочисленные – выполняются процессором, их мы и будем изучать. Нецелочисленные – математическим сопроцессором, за рамками курса.

**Целочисленные команды:**

Поскольку Assembler отличается от других языков тем, что все его команды элементарны, и для выполнения любого действия требуется написать в программе целых блок команд, основную часть программы составляют комментарии. Иначе вы уже через 5 минут забудете, что вы имели ввиду раньше. Комментарий начинается с символа **;** и продолжается до конца строки.

**1) Пересылка данных:**

**MOV AX, BX**            ; AX = BX - присвоение значения (после **;** идет комментарий до конца строки)

                                   ; AX - приемник (первым всегда имя переменной или регистра - туда сохр. результат)

                                   ; BX - источник (может быть как имя, так и значение - число)

**MOV AH, 11**             ; AH = 11 – присвоить число 11 в десятичной системе

**MOV AL, 11h**           ; AL = 1116 – присвоить число 11 в шестнадцатиричной системе (=1710)

**MOV BX, 002Fh**      ; BX = 2F16 - необходимо следить за совпадением разрядности операндов

**MOV CX, 1111000011110000b**  ; CX = F0F016 – пример присвоения двоичных чисел

**XCHG AX, BX**         ; меняет значения местами (оба должны быть одинаковой разрядности)

**PUSH AX**                 ; помещает AX в стек  (**PUSH AL** – нельзя, почему? *…элемент стека - 2 байта*)

**POP AX**                    ; извлекает слово из стека и помещает его в AX

**Примечание:**если требуется записать 16-ричное число, начинающееся с буквы (F4CD), то будет сообщение об ошибке. Дебаггер подумает, что это имя переменной и не найдет такого. Поэтому надо поставить старший ноль (0F4CD), тогда все сработает как надо.

**2) Арифметические операции (двоичная и десятичная арифметики):**

**ADD AX, BX**; AX = AX + BX  - сложение (результат всегда в первом операнде - приемнике)

**INC   AX**                   ; AX = AX + 1   - инкремент

**SUB AX, BX**            ; AX = AX - BX  - вычитание (результат всегда в первом операнде - приемнике)

**DEC AX**                    ; AX = AX - 1   - декремент

**NEG DX**                    ; переводит в дополнительный код и обратно (смена знака числа)

**3) Логические операции (манипуляции с битами):**

**NOT AX**                     ; инверсия (поразрядное отрицание); зная ее мы можем написать замену команде **NEG AX = NOT AX + INC AX** (но не будем, т.к. NEG - выгоднее)

**MOV AX, 0097h**      ; присвоим в AX начальное значение     0000 0000 1001 01112

**AND  AX, 00F0h**      ; поразрядное наложение на AX  маски 0000 0000 1111 00002

0000 0000 1001 0111 – значение в AX  (в командах нам будет удобнее задавать 16-ричные числа)

0000 0000 1111 0000 – маска (так что не забываем правила быстрого перевода **X2 ↔ Y16**)

--------------------------------

0000 0000 1001 0000 (в AX обнуляются биты, где в маске стоят нули, прочие - без изменений)

**MOV AX, 0097h**      ; присвоим в AX начальное значение    0000 0000 1001 01112

**OR    AX, 00F0h**      ; поразрядное дополнение AX  маской  0000 0000 1111 00002

0000 0000 1001 0111 – значение в AX  (из прошлого примера)

0000 0000 1111 0000 – маска                 (из прошлого примера)

--------------------------------

0000 0000 1111 0111 (в AX единицы в тех битах, где в маске стоят единицы, прочие - без изменений)

**MOV AX, 0097h**      ; присвоим в AX начальное значение    0000 0000 1001 01112

**XOR  AX, 00F0h**      ; шифрование AX по маске                     0000 0000 1111 00002

0000 0000 1001 0111 – значение в AX

0000 0000 1111 0000 – маска (чтобы расшифровать число, нужно наложить ту же маску еще раз)

--------------------------------

0000 0000 0110 0111 (в AX инвертируются биты, где в маске стоят единицы, прочие - без изменений)

**SHL AX, 1**                ; сдвиг AX на 1 разряд влево (умножение на 2, старшие могут ”потеряться”)

**SHR AX, 3**                ; сдвиг AX на 3 разряда вправо (деление нацело на 8, младшие - отбрасываются)

**TEST AX, 0103h**    ; проверяет те разряды AX, которые указаны в маске 0000 0001 0000 0011

                                 ; ZF = 0 если в AX есть хоть одна единица среди указанных, ZF = 1 если ни одной

**CMP  AX, BX**           ; анализирует разность (AX - BX), устанавливает флаги ZF и SF

                                  ; флаг знаков: SF = 0 если AX-BX≥0, SF = 1 если AX-BX<0   (0 – ”**+**”, 1 – ”**-**”)

                                  ; флаг значений: ZF = 0 если AX≠BX, ZF = 1 если AX=BX   (разность = или ≠ 0)

Примечание: команды **TEST** и **CMP** только меняют флаги, значения операндов остаются прежними!

Пример взаимозаменяемости команд: задача - обнулить AX

            1. **MOV AX, 0000h**

            2. **SUB  AX, AX**

            3. **AND  AX, 0000h**

            4. **XOR  AX, AX**          ; наиболее быстрый способ (без лишней пересылки данных)

            5. **SHR  AX, 16**

**4) Передача управления (безусловный и условный переход):**

Мы привыкли, что в языках высокого уровня использование операторов перехода является дурным тоном. В Assembler-е же любое разветвление алгоритма (отклонение от линейности) реализуется только через подобные операторы!

**METKA1:      ........**

**JMP METKA1**         ; безусловный переход к указанной метке (вверх или вниз)

Переход по условию:

**JS     METKA**           ; переход происходит, если SF = 1 (буква **S**в имени команды – указывает флаг)

**JNS  METKA**           ; переход происходит, если SF = 0 (буква **N**в имени команды – как бы отрицание)

**JZ     METKA**           ; переход происходит, если ZF = 1 (буква **Z**в имени команды – указывает флаг)

**JNZ  METKA**           ; переход происходит, если ZF = 0 (буква **N**в имени команды – как бы отрицание)

Цикл с параметром (CX – используется по умолчанию в качестве счетчика):

**MOV CX, 5**; установим количество повторов цикла

**METKA2:      ............................**

**… тело цикла ...**; блок команд, которые надо повторить в цикле (регистр CX - занят)

**............................**

**LOOP METKA2**       ; если CX≠0, то уменьшает CX на 1 и возвращается вверх к метке

**5) Цепочечные (выполнение действий над заданной цепочкой данных):**  пропускаем

**6) Управление состоянием ЦП:**

**INT***<номер>*           ; вызов прерывания (когда обращаемся к какой-либо стандартной функции)

Пример:

; [=== Стандартное завершение программы ===]

**MOV AH, 4Ch**          ; номер функции из библиотеки данного прерывания, которую мы хотим вызвать

                                    ; **4C** - передача управления родительской задаче (т.е. операционной системе)

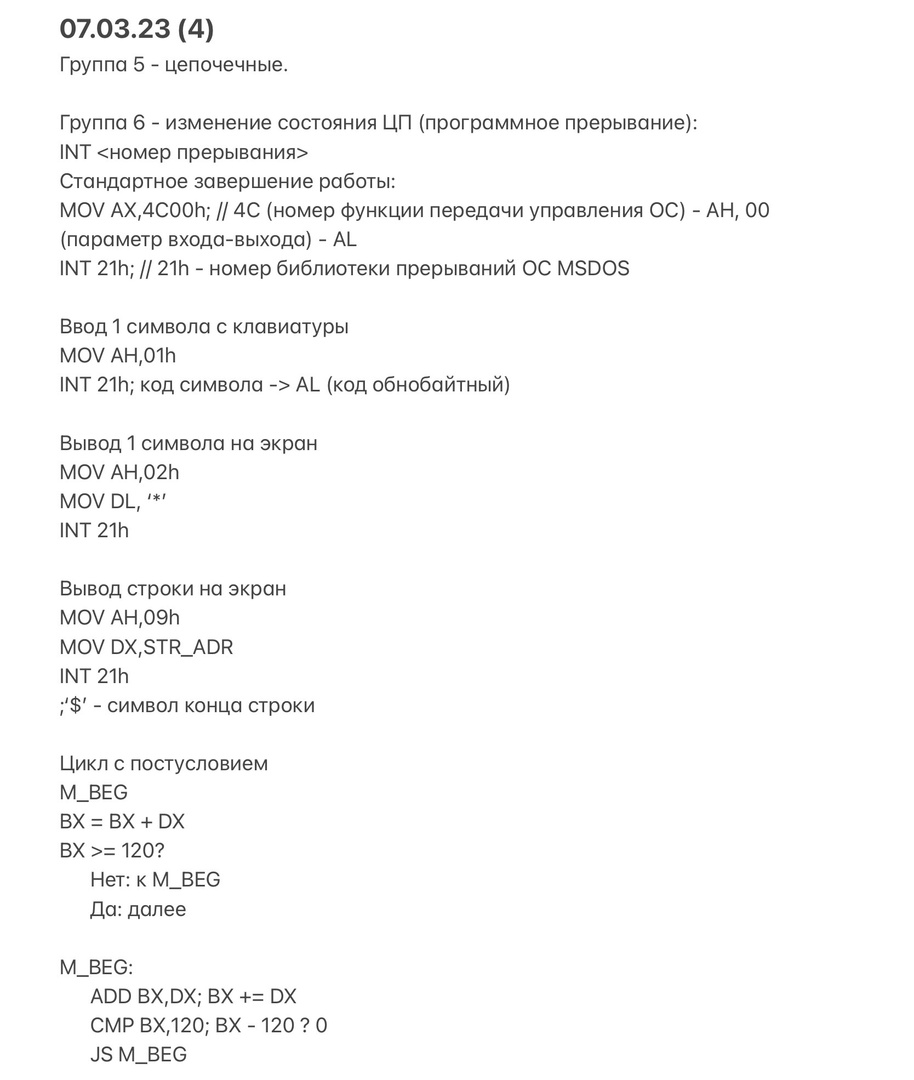
**MOV AL, 00h**           ; передаваемый в функцию параметр (выход с кодом **0**, т.е. без ошибки)

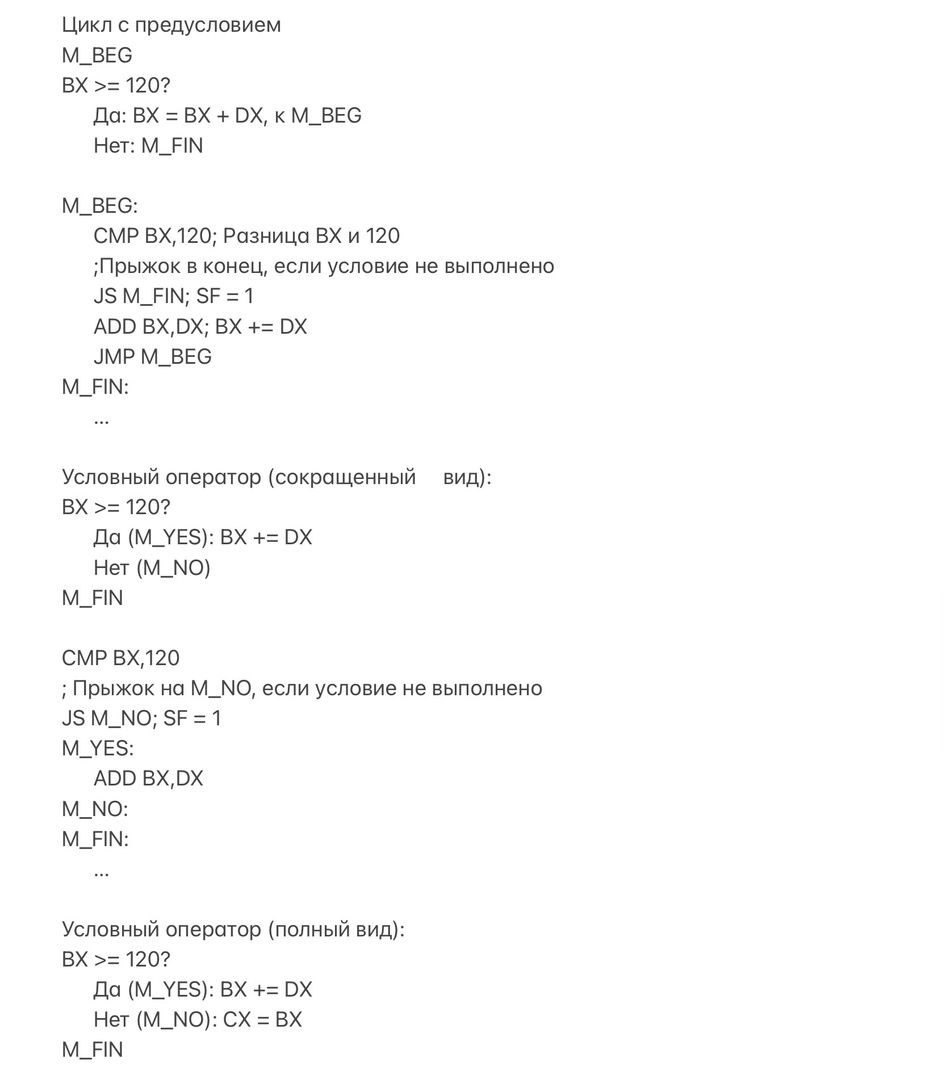
                                   ; можно было сразу присвоить **MOV AX, 4C00h**, но в две строки вам нагляднее

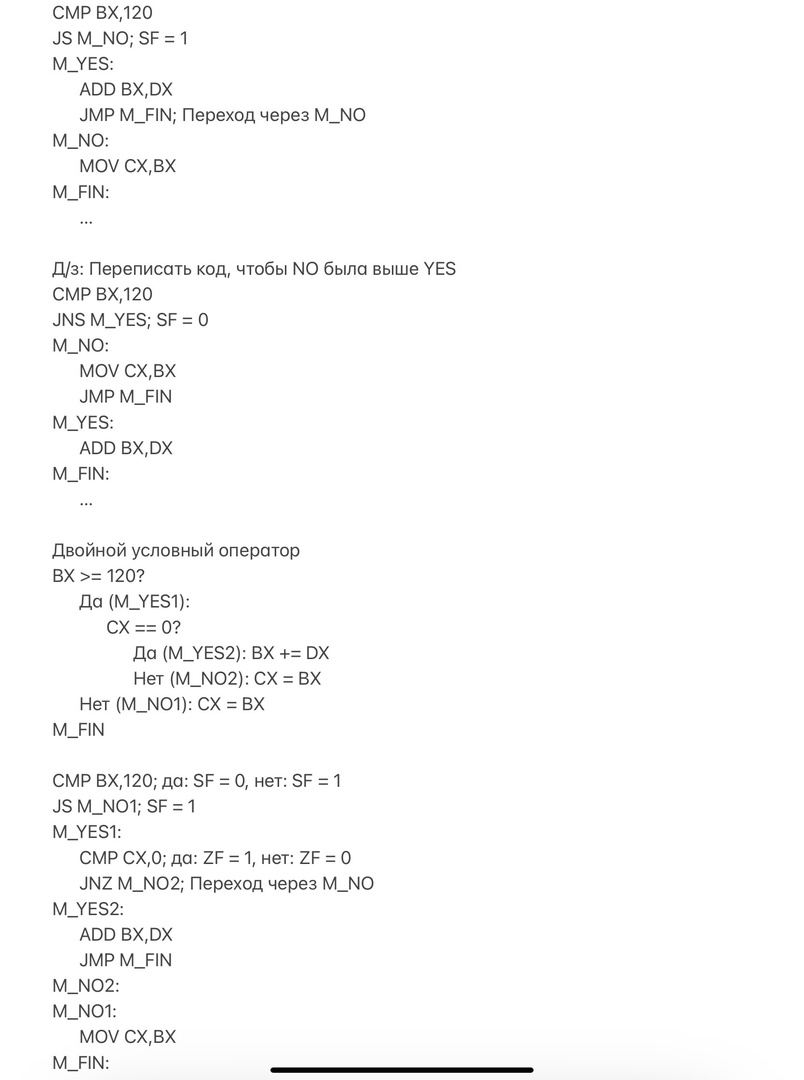
**INT   21h**                   ; вызов прерывания DOS – выполняется указанная функция (завершение работы)

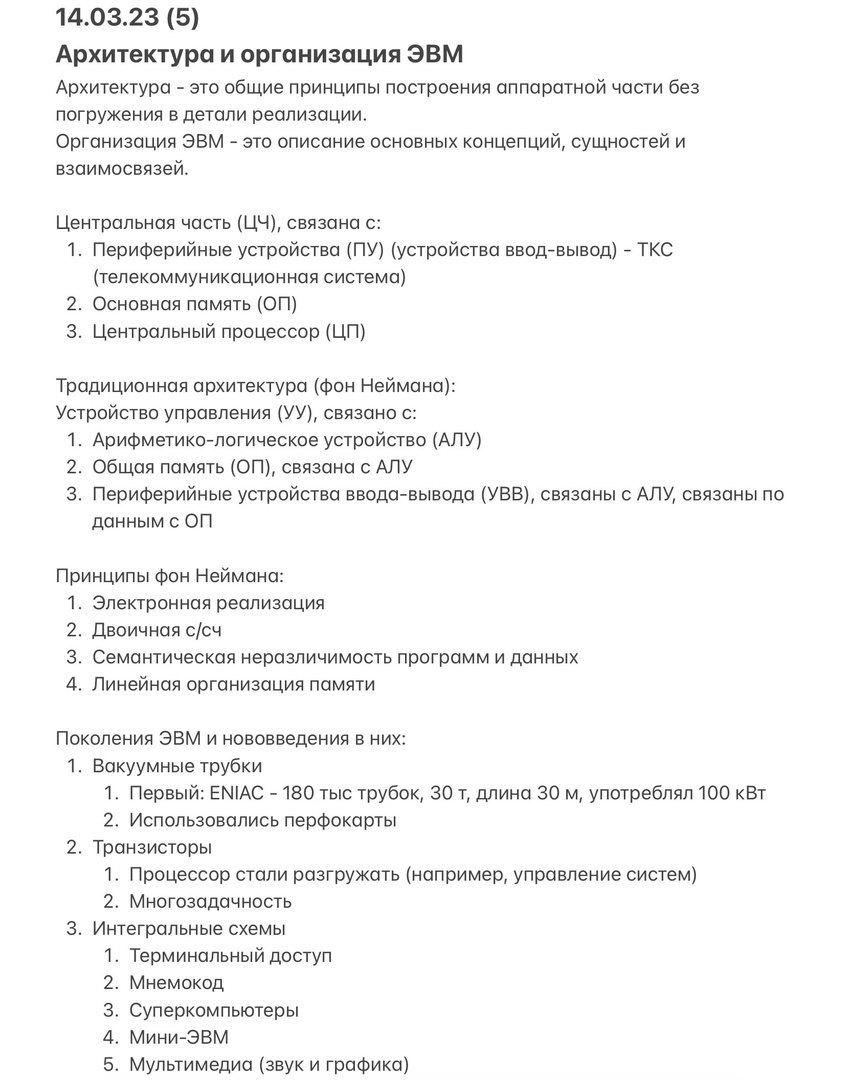
Если в конце программы не поставить такой блок команд, то дебаггер не узнает, что программа завершена, и продолжит считывать из памяти и выполнять всякий ”мусор”!

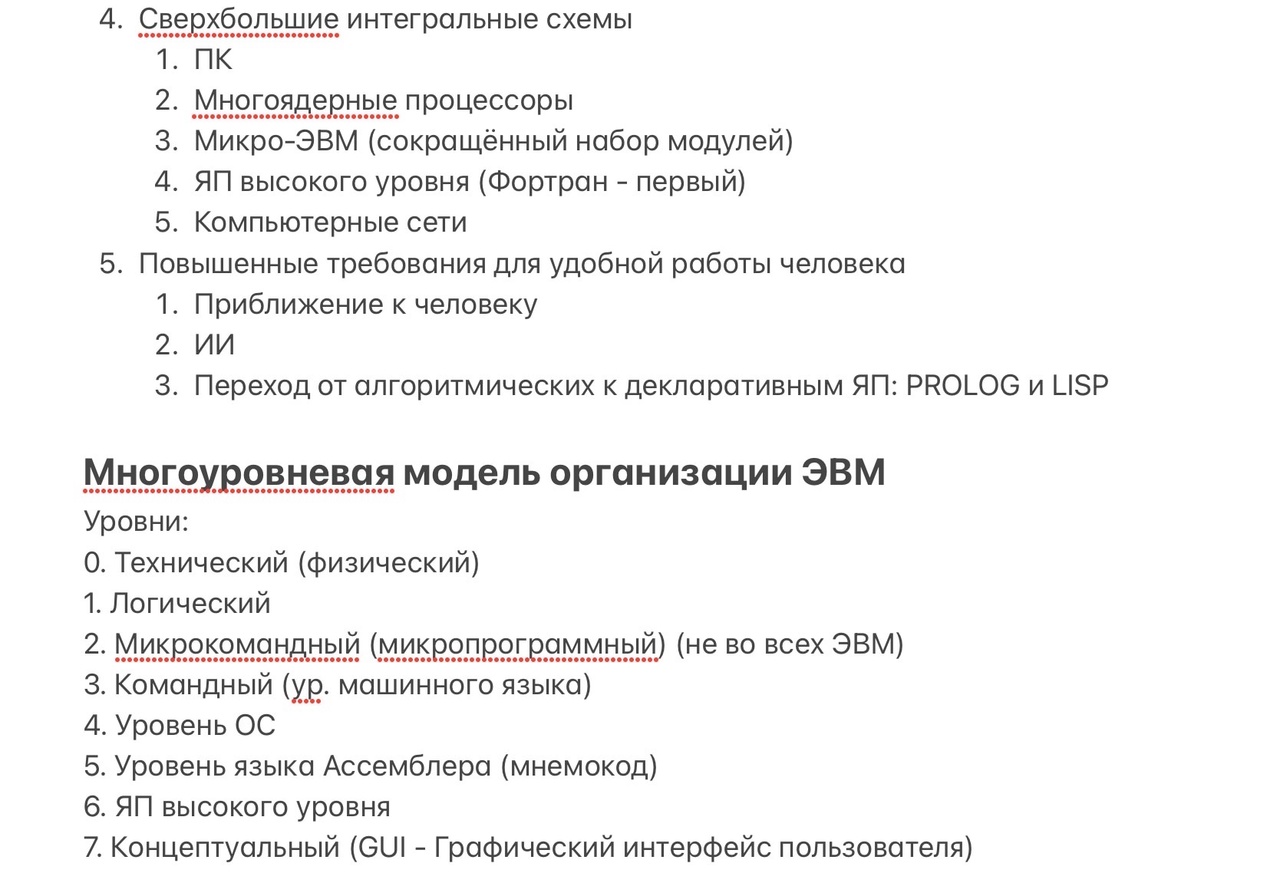
Как видите, команда **INT** всегда обращается к регистру **AX**, чтобы узнать исходные данные вызова – номер библиотечной функции, передаваемый параметр. Кроме того, некоторые функции будут обращаться за данными к регистру **DX** и совсем редко – к **BX**, но об этом позже.











\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ПРОПУСК\_ПРЕД\_ЛЕКЦИИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

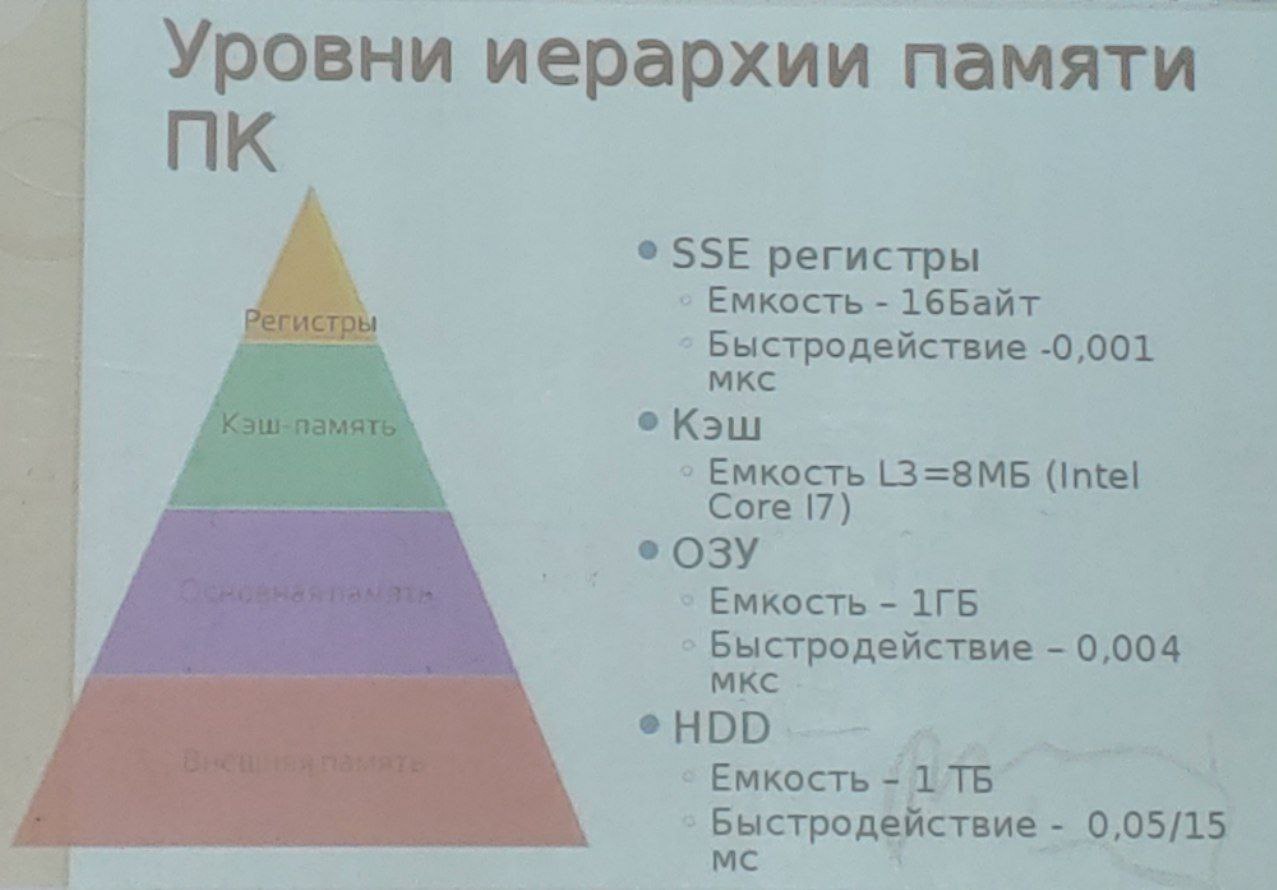
**Иерархическая структура памяти ЭВМ.**

Начиная со в-го поколения ЭВМ начался переход к иерархической структуре памяти. Запоминающие устройства (ЗУ) разных типов обладали весьма разными характеристиками (объем – машинные слова(килослова, мегаслова,…), количество байт, которое может обработать ЭВМ за один раз), быстродействие – операции в секунду(целочисленные vs с плавающей запятой), Цикл доступа = 1/Быстродействие(нс) – доступ на чтение и доступ на запись, удельный объем = объем / объем носителя(байт/мм^3) – устаревшая характеристика, удельная стоимость - стоимость в пересчете на единицу хранимой информации.

0.123...Е12

0.12… - мантисса, 12 – порядок.

Использование самого лучшего типа памяти экономически невыгодо.



КЭШ – реализуется на триггерах(как и регистры), адресуется, имеет больший объем, позволяет разгрузить ОЗУ, хранит массивы данных. Чтение и запись осуществляется постранично. Постраничное чтение из ОЗУ и запись в кэш быстрее чем постоянное обращение к ОЗУ. Часто кэш – многоуровневый. Уровни обозначаются L1, L2, L3, уровни различаются объемом и быстродействием. Кэш память также энергозависима. Наращивание теоретически возможно.

ОЗУ - легкая масштабируемость, система адресации и система регенерации для подзарядки конденсаторов, энергозависимость.

ВЗУ – энергонезависимая память для хранения больших объемов памяти.