Весна 2022

# Системное программное обеспечение

Онлайн-лекции

## Лекция №1: **Введение**

Доцент, к.т.н. ГОЛЬЦОВ Александр Геннадьевич

# Что нам предстоит:

- Лекции каждую неделю
- Лабораторные работы (6 шт. / 7 занятий)
- Расчетное задание
- Зачет по итогам ЛР+РЗ  (экзамен отменен)

Преподаватели:

Доц. Гольцов А.Г. (ЛК, ЛР в 12 гр.)

??? (ЛР в 7 и 8 гр.)

# Предмет изучения

Язык ассемблера для процессора i8086/88 (286, 386).

Разрабатываемые программы предназначаются для запуска на компьютере класса IBM PC AT, находящегося под управлением MS DOS. Операционная среда эмулируется на современной технике.

Курс является дополнением к важнейшему курсу бакалавриата каф. ВМСС "Микропроцессорные системы".

# Литература

1. Абель П. Язык ассемблера для IBM PC и программирования. – М.: «Высшая школа», 1992. – 447 с.
2. Белецкий Я. Энциклопедия языка Си. – М.: «Мир», 1992. – 687 с.

+ любые практически книги по процессорам семейства x86 (как справочники по системе команд)

(IBM PC assembler language and programming)

# Лабораторки

- 1-я вводная, с помощью преподавателя и  без защиты, остальные – программа в электронном виде (без отчета) и беседа по ней
- Инструменты:
  - Turbo Assembler (Borland)
  - Turbo Debugger или другой отладчик по выбору студента
- Мы работаем в командной строке DOS; для 64-разрядных версий Windows необходимо использовать DosBox.

# Термин "ассемблер"

- Ассемблеры - класс инструментальных программ, разновидность трансляторов.
- А. осуществляет трансляцию с языка низкого уровня (языка ассемблера, ассемблерного языка) в бинарные файлы, содержащие машинные коды.
- Это НЕ имя собственное, пишется с маленькой буквы.
- "Написать на Ассемблере" = технический жаргон, в такой формулировке - с большой буквы.
- Для каждого семейства процессоров - свой язык ассемблера. Мы изучаем ЯА процессора i8086.

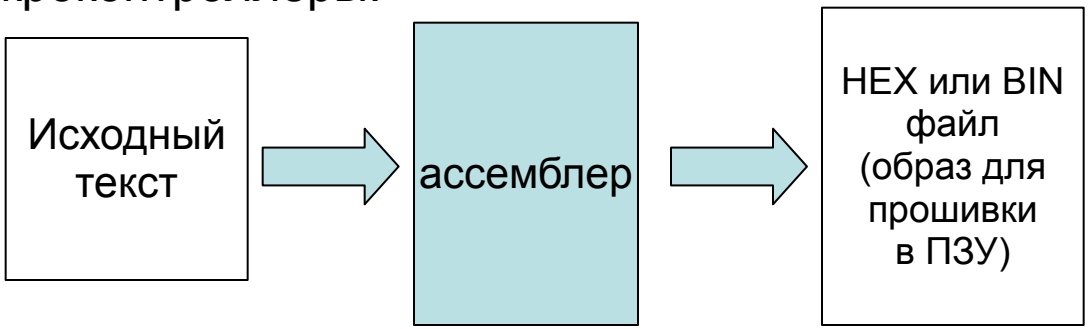
# Назначение языка ассемблера

- Описывать последовательность машинных команд, непосредственно исполняемых процессором
- Описывать структуру областей памяти
- Описывать макроопределения (данные и код)
- Описывать параметры процедур
- Описывать параметры, нужные для компоновки исполнимого файла

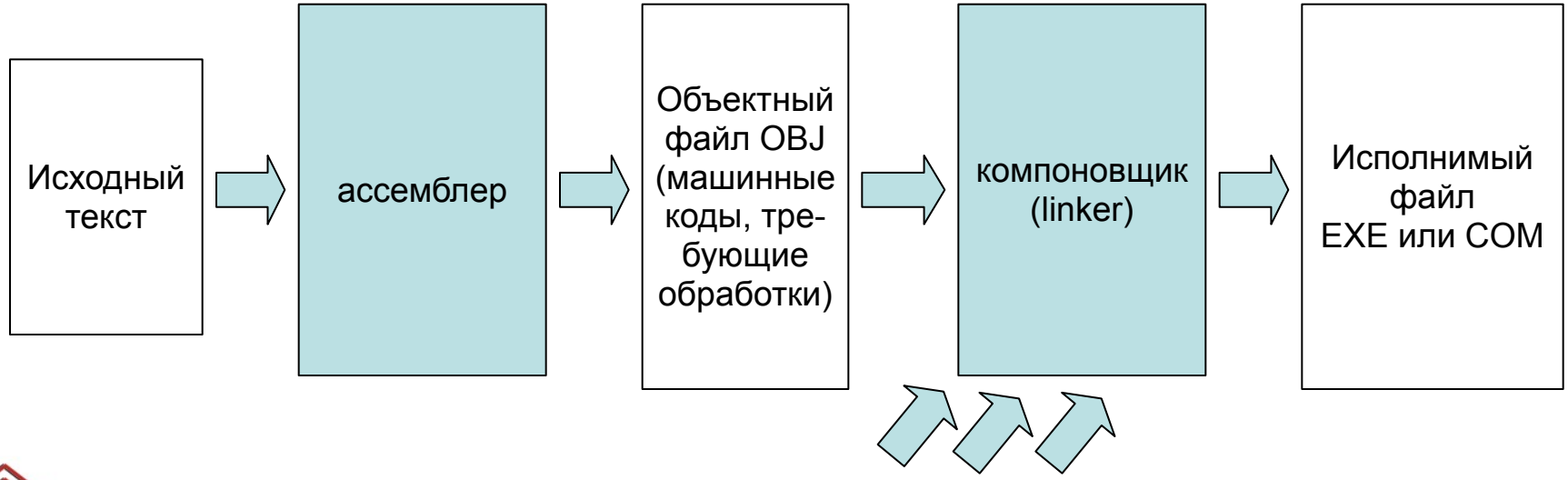
# Использование ассемблера



Микроконтроллеры:




Компьютер:



# Роль языка ассемблера в современном мире

- Для настольного компьютера на чистом Ассемблере программы практически не пишутся ➔ Си
- Для микроконтроллеров - только мелкие проекты
- На языке ассемблера пишутся отдельные процедуры и модули, компонуемые вместе с программой на ЯВУ
- Трансляторы ЯВУ поддерживают ассемблерные вставки - куски кода на подмножестве яз. ассемблера внутри программы на Си/Паскале/...

# Эффективность кода

- На языке ассемблера описывается непосредственно последовательность машинных команд.
- Грамотный программист составит максимально эффективный код.
- Написание большой программы требует высокой квалификации и чрезвычайно трудоемко, в т.ч. в отладке.
- Вполне вероятно, что код, выданный транслятором ЯВУ, окажется эффективнее, чем то, что напишет программист недостаточной квалификации на Ассемблере.

# Синтаксис

- Используются символы английского алфавита
- Синтаксическая единица - строка
- Одна строка - одна машинная команда или директива языка ассемблера
- Метки
- Комментарии - после точки с запятой до конца строки
- Строки - в одиночных или двойных кавычках
- Числа - в десятичной, двоичной, шестнадцатеричной системе счисления

# Команды и директивы

- Команда - мнемоника и описание параметров, непосредственно преобразуемые в машинный код одной машинной команды, который аппаратно исполняет процессор.

```
mov ax,25   ; занести число 25 в регистр AX
```

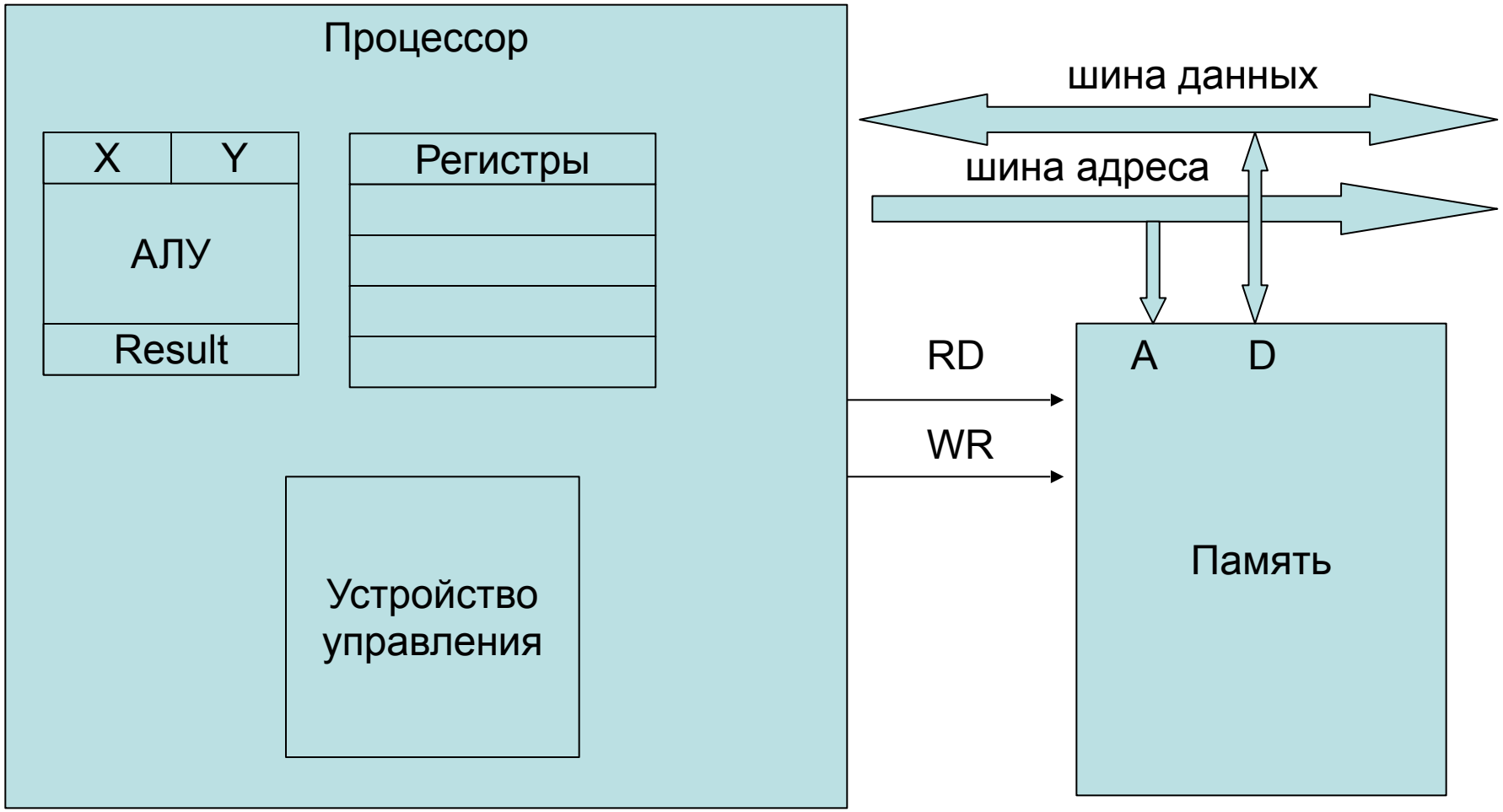
- Директива - фраза языка, НЕ преобразуемая в машинный код. Например - управляющая процессом компиляции или задающая модель памяти, атрибуты сегмента...

# Чего нет в Ассемблере

- Переменных.
- Операторов присваивания.
- Операторов ветвления, цикла.
- Операторов ввода-вывода.
- Готовых библиотек, содержащих какие-то даже простейшие функции (математические, преобразование чисел в строки и т.п.)

Все подобные конструкции реализуются при помощи поддерживаемого процессором набора машинных команд.

# Как процессор исполняет программу



# Исполнение программы

- Адрес текущей команды хранится в регистре, называемом обычно **счетчиком команд**
- Процессор считывает байты с адресов, начиная с адреса в СК, пока не соберется последовательность байтов, представляющая машинную команду
- СК при этом увеличивается на длину считанного машинного кода и устанавливается на начало следующей команды
- Переход (условный, безусловный, вызов процедуры, возврат из процедуры) означает принудительное изменение значения СК на другое, отличное от адреса следующей команды.

# Машинный код

- Команда занимает от 1 до 5 и более байт
- Состоит из
  - кода операции,
  - модификаторов (кодируют тип параметров)
  - параметров

| КОП | Мод | пар.1 | пар.2 |
|---|---|---|---|

int 21h ➔ CD 21

cli ➔ FA

mov ax,2048d ➔ B8 00 08

jmp 1234h:5678h ➔ EA 78 56 34 12

```
; общий комментарий - автор, особенности запуска и т.п.
.model small              ; один сегмент кода, данных и стека
.stack 100h               ; отвести под стек 256 байт
.data                     ; начало сегмента данных
S  db   'Hello, world!$'
.code                     ; начало сегмента кода
   ; Начальная инициализация
   mov  ax,@data
   mov  ds,ax             ; настройка DS на начало сегмента данных

   ;-----------------------------------------------
   ; Вывод строки на экран
   mov  ah,9              ; номер функции DOS
   mov  dx,offset S       ; DS:DX <- адрес строки S
                          ; DS уже проинициализирован ранее
   int  21h               ; Вывод строки на экран в текущей позиции
   курсора
   ;-----------------------------------------------

   ; Стандартное завершение программы
   mov  ax,4C00h          ; ah = N функции, al = код возврата
   int  21h               ; снять программу с выполнения

   end          ; конец текста программы
```

# Компиляция программы

- Имеем исходный текст my.asm
- Текст в кодировке ASCII, не UTF-8, не документ Word
- Ассемблирование:
  tasm my.asm ➔ my.obj (или ошибки)
- Сборка:
  tlink my.obj ➔ my.exe (или ошибки)
- Запуск:
  my.exe
- Отладка:
  td my.exe

# Стиль оформления кода

- Метки - слева, команды - через табуляцию
- Комментарии необходимы
- Многоуровневые комментарии, необходимость убывает с уровнем:
  - на всю программу
  - на процедуру (что делает, какие параметры)
  - на блок команд (что эти несколько команд делают в целом)
  - на каждую команду

```
xor ax,ax  ; исключающее или ax и ax
```

```
xor ax,ax  ; обнулить ax
```

# Метки

- Метка состоит из букв, цифр и подчеркиваний
- Метка начинается с буквы или подчеркивания
- После метки ставится двоеточие, если эта метка в коде
- Большие и малые буквы не различимы
- Если метка используется при определении данных - двоеточие не ставится
- При ссылке на метку двоеточие не ставится
- Перед локальными метками ставится @ или другой символ (рассмотрим, когда дойдет дело)

```
m1: .....
    .......
    jmp m1
```

```
_str db 'Hello!'
                ..................
mov dx, offset _str
```

# Числа

- Начинаются с цифры от 0 до 9
- Могут заканчиваться модификатором системы счисления:
  H = шестнадцатеричная
  D = десятичная (по умолчанию)
  B = двоичная
- В шестнадцатеричной системе буквы ABCDEF используются в качестве цифр 11, 12, 13, 14 и 15.
- Допустимы большие и малые буквы
- Правильные числа:
  1000, 1000B, 1000h, 1e8h, 0FFFh
- Неправильные числа:
  1B2D (десятичное?), E000 (это метка), 25CB (двоичное?)

# Шестнадцатеричные числа

- Нужны для сокращенной записи двоичных
- Одна тетрада в двоичной записи ➔ одна шестнадцатеричная цифра
- Каждый студент ВМСС должен уметь в уме легко и непринужденно переводить из двоичной системы в шестнадцатеричную и наоборот, никаких делений в столбик и выписывания остатков!!!
- 0000 = 0
  0001 = 1
  0010 = 2
  0011 = 3
  0100 = 4
  0101 = 5
  0110 = 6
  0111 = 7
  1000 = 8
  1001 = 9
  1010 = A
  1011 = B
  1100 = C
  1101 = D
  1110 = E
  1111 = F
  1001 1110 = 9E

# Обращение к памяти

- Шина адреса 20-разрядная ➔ 1M памяти
- Регистры 16-разрядные
- 20-разрядный адрес на шине формируется из двух 16-разрядных компонентов - сегмента и смещения
- Сегментная часть адреса при обращении к памяти заранее заносится в сегментный регистр, смещение указывается в команде явно или тоже задается в регистре.
- Сегментная часть умножается на 4 и складывается со смещением:
  1234:5678h ➔ 179B8h (выставляется на шине)

```
 12340
+ 5678
 179B8
```

```
mov ax, 1234h
mov es, ax
mov ax, es:[5678h]
```

# Команды и операнды

- В команде может не быть операндов:
  cli  std  movsb
- В команде может быть один операнд
  int 21h  push ax  not al
- В команде может быть два операнда
  mov ax,bx  add al,[si]
- Больше двух операндов в 8086 нет
- Операнды задают данные: непосредственно, прямо, косвенно, в регистрах
- Часть данных может задаваться неявно
  push ax ::  ax → mem(ss:sp);  sp+2 → sp
- Если два операнда, то результат в первом операнде

# Виды адресации

- Непосредственная: сам операнд прописан в коде команды:
  mov ax, 20
- Регистровая: операнд находится в регистре:
  mov ax, 20
- Прямая: адрес (смещение) операнда прописано в коде команды:
  mov ax, [20]  ; mem(ds:20) -> AX
- Косвенная: адрес (смещение) операнда задан в регистре:
  mov ax, [bx]  ; mem(ds:bx) -> AX

# Регистры i8086

- Все регистры 16-разрядные
- Общего назначения (РОН)
  - можно манипулировать в программе относительно свободно
  - имеют каждый особенности использования в системе команд
- Специального назначения (РСН)
  - управляют работой процессора, адресацией, выборкой команд
  - прямое изменение значений невозможно или легко приводит к катастрофическим последствиям для системы

# РОН i8086

- AX, BX, CX, DX
  - отдельно доступны старшие и младшие байты:
    AH и AL, BH и BL, CH и CL, DH и DL
  - **AX - аккумулятор**, жестко задан в некоторых командах как источник или приемник данных
  - **BX - указатель базы**, может использоваться при косвенной адресации к ячейкам памяти
  - **CX - счетчик**, часто используется для задания количества повторений действия
  - **DX - расширение аккумулятора**, используется для хранения старшего слова 32-разрядных чисел при умножении и делении и в качестве адреса порта при косвенном обращении к портам
- SI, DI - индексные регистры
  - **SI = source index**, смещение в памяти, откуда берутся данные при использовании строковых команд (полный адрес DS:SI)
  - **DI = destination index**, смещение в памяти, куда помещаются данные при использовании строковых команд (полный адрес ES:DI)

# Сегментные регистры i8086

- CS, DS, ES, SS - сегментные регистры
  - используются при формировании адресов, содержат сегментную часть адреса
  - часто используются неявно
  - CS - сегмент кода, из которого выбираются команды
  - DS - сегмент данных, из которого выбираются данные по умолчанию
  - ES - дополнительный сегментный регистр для обращения к данным или для строковых команд
  - SS - сегмент стека

# Регистр IP

- CS:IP всегда содержит полный адрес текущей команды, код которой исполняется процессором
- Изменение CS или IP по любой причине означает переход, т.е. следующая по порядку команда не исполняется и выборка команды происходит из места, куда теперь указывает CS:IP
- Команды JMP, CALL, RET, INT, IRET и др. изменяют значение CS:IP
- Явное изменение CS и IP не предусмотрено

# SP и организация стека

- SP - указатель стека
- В стек заносятся двухбайтовые слова ➔ SP четный
- "Нарастает вниз": уменьшается на 2 при занесении значения в стек, увеличивается на 2 при извлечении из стека
- SS:SP - адрес последней занятой ячейки в стеке
- Если стек пустой - SS:SP указывает за пределы сегмента стека на следующее за его верхней границей слово

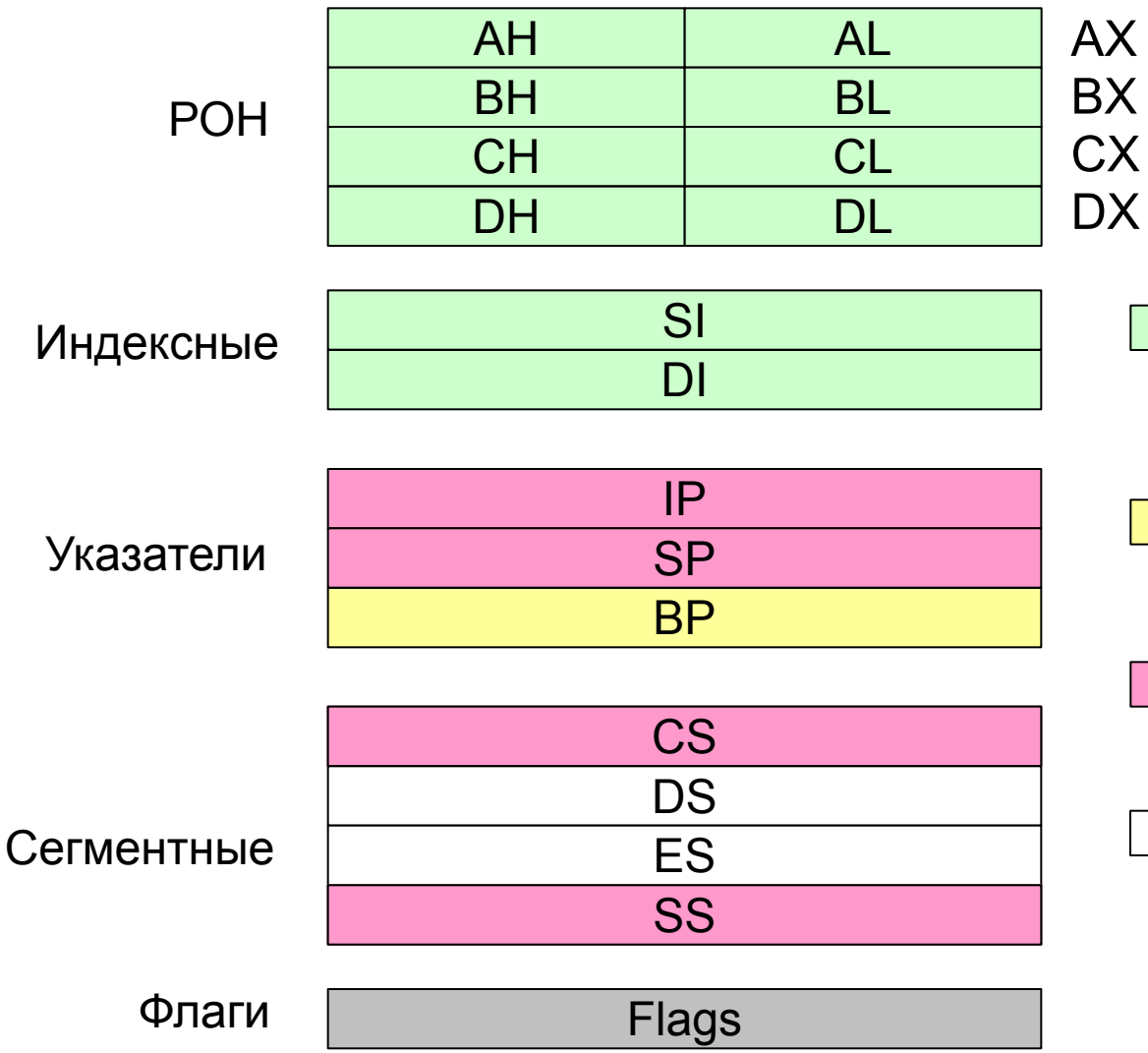
# Регистр BP

- **BP** обычно используется для косвенного обращения к ячейкам памяти в сегменте стека
- Используется в коде, генерируемом ЯВУ, в качестве адреса начала стекового кадра процедуры
- В чисто ассемблерной программе можно использовать относительно свободно
- При использовании в качестве косвенного адреса - по умолчанию сегментная часть берется из SS, а не из DS, как для регистров BX, SI, DI

# Регистр флагов

- Часть битов выставляются при совершении арифметических и логических операций и характеризуют результат:
  Z - нулевой результат
  C - перенос из старшего разряда
  S - отрицательный знак числа
  O - арифметическое переполнение
  A - межтетрадный перенос (двоично-десятичные преобразования)
  P - четное количество единиц в младшем байте результата
- Другие биты управляют работой процессора:
  I - разрешение прерываний
  T - режим трассировки
  D - перебор операндов в порядке уменьшения/увеличения адресов в строковых командах
- Регистр в целом можно сохранить в стеке и восстановить PUSHF/POPF

# Регистры i8086



# Спасибо за внимание.