



НИЗКОУРОВНЕВОЕ ПРОГРАММИРОВАНИЕ

Лекция 1



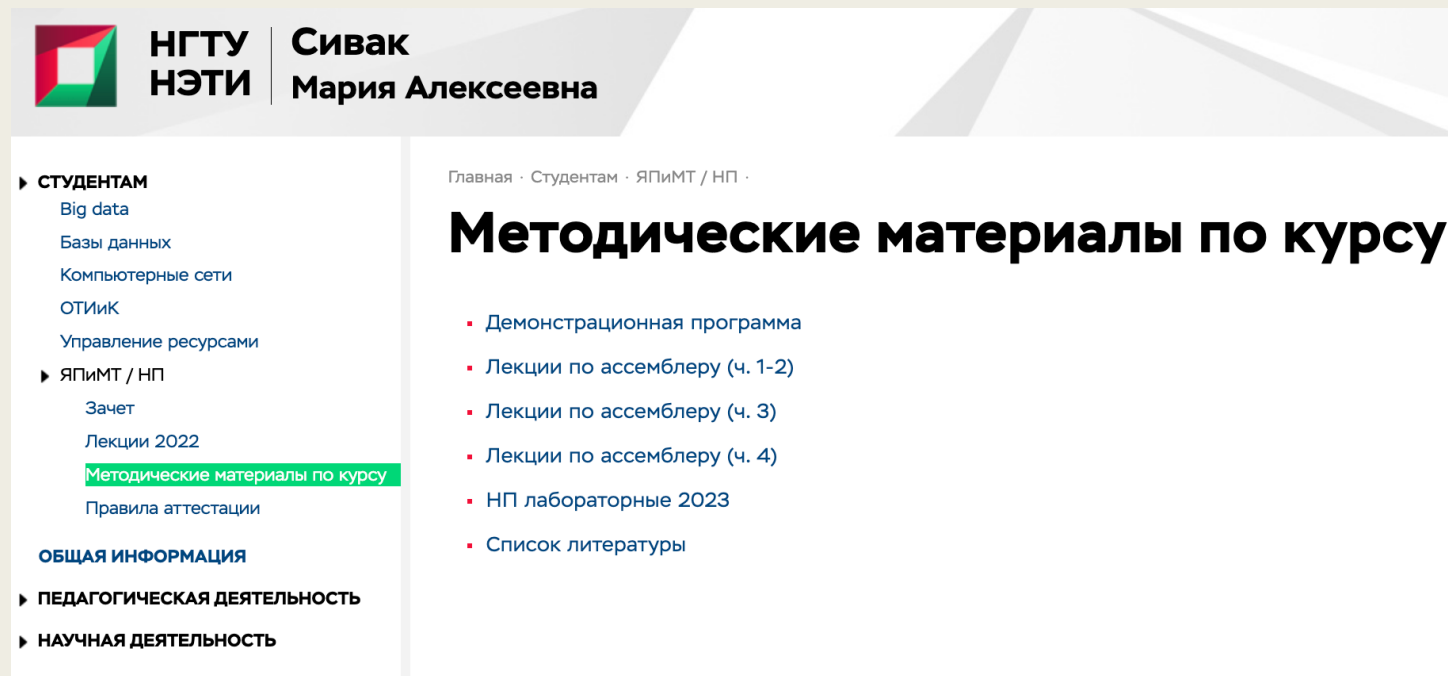
Зачем оно вам надо?

- Понимание того, как ваша программа работает на уровне «железа»
- Способность определить, не в компиляторе ли проблема
- Задача обратного инжиниринга



Структура курса

- Лекции
- Лабораторные работы



The screenshot shows the website interface for the course. At the top, there is a header with the NGTU NÉTI logo and the name of the lecturer, Maria Alekseevna Sivak. Below the header, there is a navigation menu on the left side with categories like 'STUDENTAM', 'YPIMT / NP', 'ZACHET', 'Lectures 2022', 'Methodological materials for the course' (highlighted in green), 'Rules of attestation', 'GENERAL INFORMATION', 'Pedagogical activity', and 'Scientific activity'. The main content area on the right is titled 'Методические материалы по курсу' and contains a list of items: 'Demonstration program', 'Lectures on assembly (ch. 1-2)', 'Lectures on assembly (ch. 3)', 'Lectures on assembly (ch. 4)', 'NP laboratory works 2023', and 'List of literature'.

NGТУ
НЭТИ

Сивак
Мария Алексеевна

Главная · Студентам · ЯПИМТ / НП ·

Методические материалы по курсу

- Демонстрационная программа
- Лекции по ассемблеру (ч. 1-2)
- Лекции по ассемблеру (ч. 3)
- Лекции по ассемблеру (ч. 4)
- НП лабораторные 2023
- Список литературы

STUDENTAM
Big data
Базы данных
Компьютерные сети
ОТИИК
Управление ресурсами

YPIMT / NP
Зачет
Лекции 2022
Методические материалы по курсу
Правила аттестации

ОБЩАЯ ИНФОРМАЦИЯ

ПЕДАГОГИЧЕСКАЯ ДЕЯТЕЛЬНОСТЬ

НАУЧНАЯ ДЕЯТЕЛЬНОСТЬ

В конце семестра – **ЗАЧЕТ**
(который можно получить автоматом, если сделать всё вовремя)

Структура курса

График учебного процесса: группа: ПМИ-11, семестр: 5

N	Наименование дисциплины	Часов в семестр	Лекции	Практ.	Лаб. работы	КП	КР	РГР	Контр. работы	Экз.	Зач.	Тип
1	Физическая культура и спорт	36									зач.	Обяз.
2	Теория вероятностей и математическая статистика	144	18	36						экз.		Обяз.
3	Операционные системы и компьютерные сети	108	18		16,2						зач.	Обяз.
4	Численные методы	144	18		16,2					экз.		Обяз.
5	Проектирование интерфейсов и системный анализ	108	18		18						зач.	Обяз.
6	Низкоуровневое программирование	108	14,4		16,2						зач.	Обяз.
7	Основы web-программирования	108	18		16,2						зач.	Обяз.
8	Объектно-ориентированное программирование	144	14,4		27					экз.		Обяз.
9	Теория информации и криптография	108	18		18					экз.		Обяз.
10	Учебная практика: ознакомительная практика	108									зач.	Обяз.
11	Проектная деятельность	72		27							зач.	Факульт.
12	Иностранный язык (для продолжающих обучение)	72		36							зач.	Факульт.
Всего:										4	8	

Лабораторные работы: содержимое отчета

- Задание в соответствии с номером варианта
- Используемые программные средства языка Ассемблер
- Текст программы (крайне желательно с комментариями)
- Набор тестов

Лабораторные работы: как проходит сдача и защита

- Сдача работы:
 - Демонстрация отчета (в печатном виде)
 - Демонстрация работы программы
- Защита работы:
 - Ответы на вопросы (обычно на контрольные)

Что нужно знать для начала

- семейство (архитектура) процессоров
- регистры
- компиляторы и синтаксис
- модель памяти
- исполняемые файлы
- процесс компиляции

Ассемблер VS Язык ассемблера

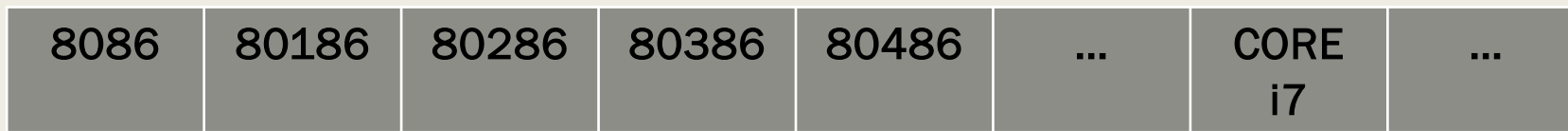
- **Язык ассемблера** – это язык программирования, состоящий из команд процессора, представленных в виде символических обозначений.
- **Ассемблер** – это компилятор языка ассемблера в машинный код.

Машинная команда	Команда на языке ассемблера
10111000	mov

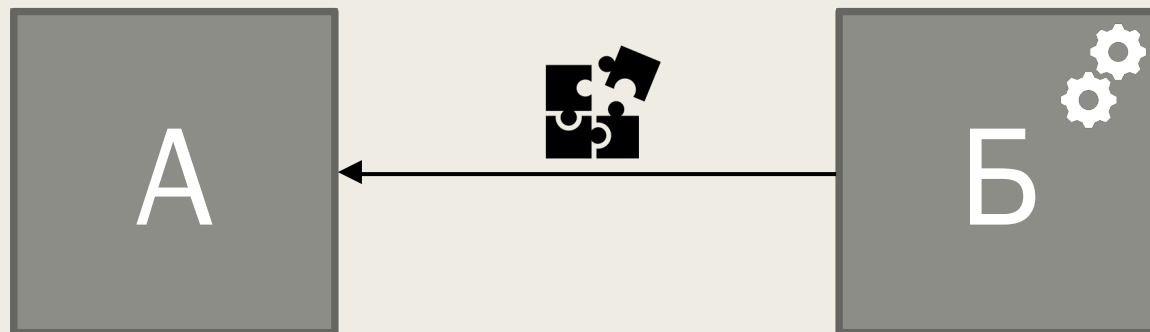
Семейство (архитектура) процессоров

Архитектура процессоров (с программной точки зрения) – совместимость с определенным набором команд, их структурой (система адресации, набор регистров) и способа исполнения (счетчик команд).

Семейство процессоров



Обратная совместимость – все программы, написанные для модели А, должны работать для модели Б.



Семейство (архитектура) процессоров

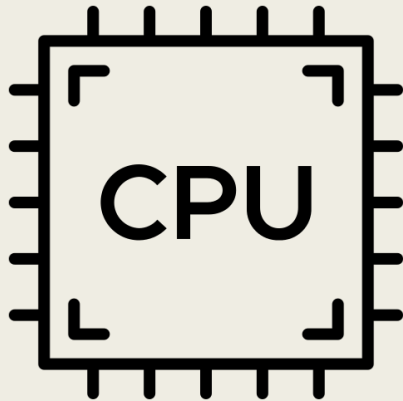
Семейство процессоров

По умолчанию



Регистры

- Регистры – маленькие ячейки памяти, расположенные на самом процессоре, в которые помещаются инструкции из памяти для дальнейшего их выполнения.



Регистры общего назначения: **AX BX CX DX**

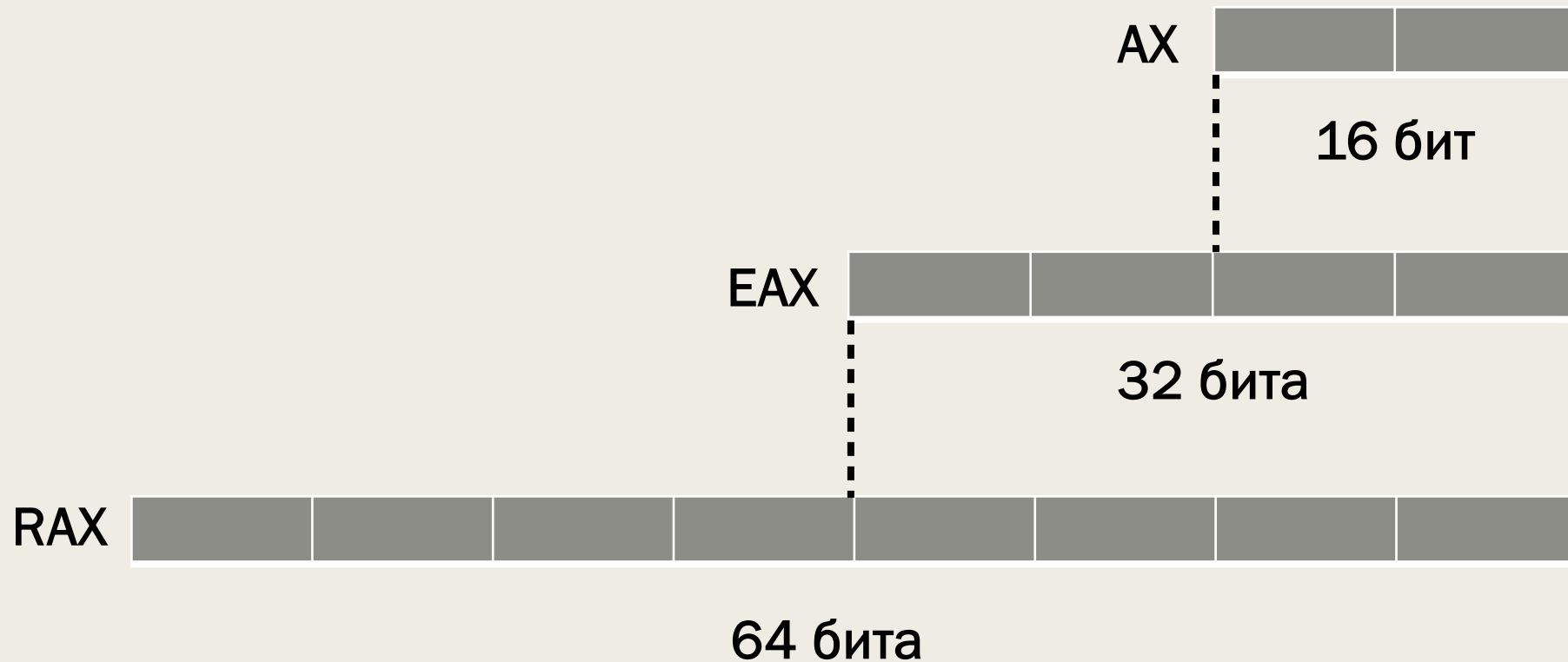
Регистры-указатели: **SP BP IP**

Индексные регистры: **SI DI**

Сегментные регистры: **CS DS SS ES GS FS**

Регистр состояния: **FLAGS**

Регистры общего назначения



Указатели и индексы

16 разрядов	32 разряда	Назначение регистра
BP	EBP	Указатель базы
SP	ESP	Указатель стека
IP	EIP	Указатель команд
SI	ESI	Индекс источника
DI	EDI	Индекс приёмника

*Регистр IP/EIP (указатель команд) содержит смещение следующей подлежащей выполнению команды, единственный регистр, который недоступен программисту. Иногда его называют регистром управления.

Сегментные регистры

16 разрядов	Назначение регистра
CS	Сегментный регистр кода
DS	Сегментный регистр данных
SS	Сегментный регистр стека
ES, GS, FS	Дополнительные сегменты данных

Регистр состояния FLAGS

- Содержит в себе флаги, которые делятся на 3 группы:
 - **Флаги состояния** – отражают особенности результата исполнения арифметических или логических операций
 - **Флаг направления** – используется цепочечными командами, определяет направление обработки цепочки
 - **Системные флаги** – управляют вводом/выводом, маскируемыми прерываниями, отладкой, переключением между задачами и виртуальным режимом 8086.

Компиляторы ассемблера

INTEL

`mov ax, 3`
`mov ax, bx`

`mov` команда пересылки данных

`op1` куда копируем

`op2` откуда копируем

AT&T

`movw %ax, %bx`

b (byte) – операнды размером в 1 байт

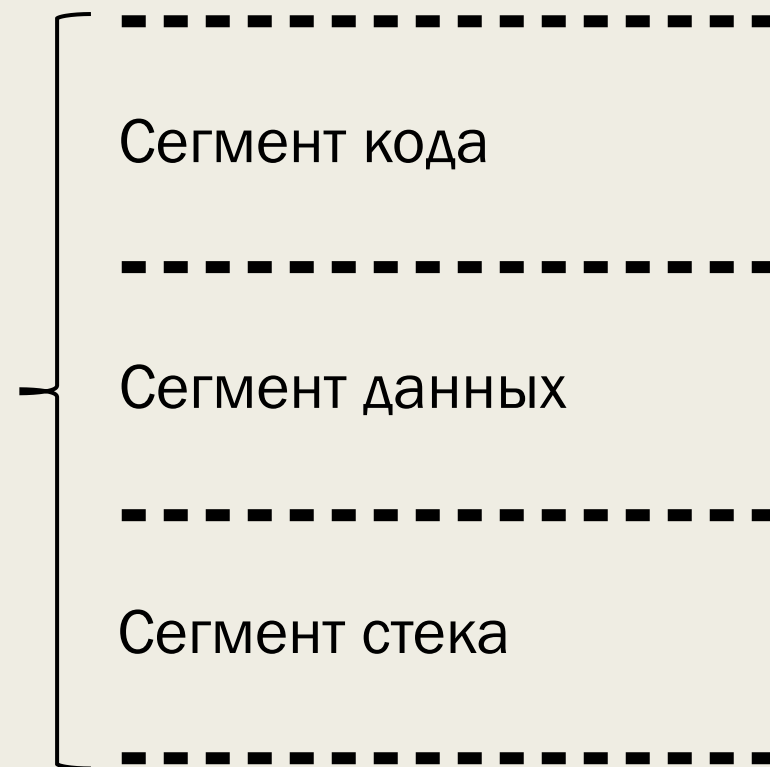
w (word) – операнды размером в 1 слово

l (long) – операнды размером в 4 байта

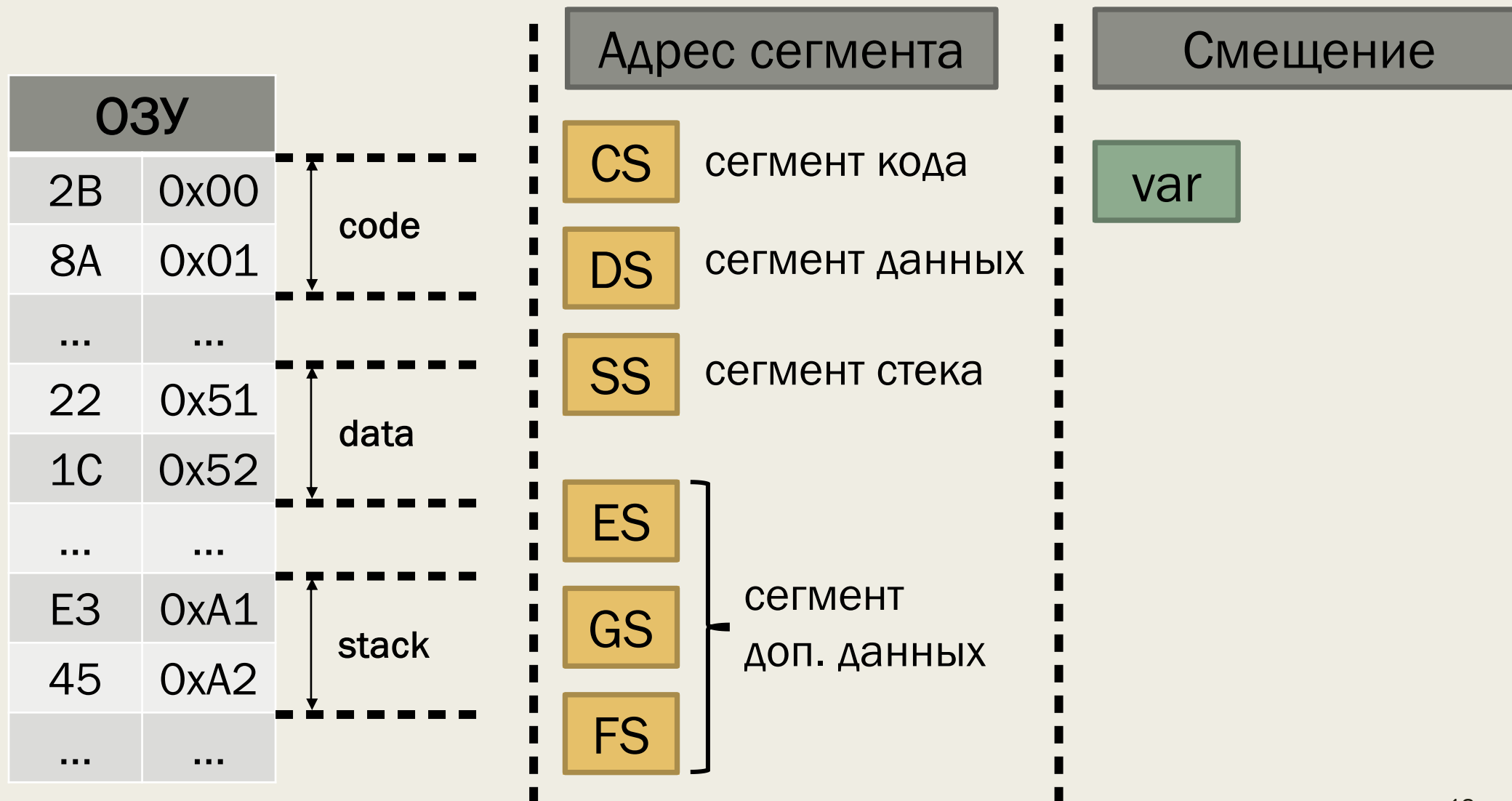
Сегменты программы

ОЗУ		
2B	0x00	code
8A	0x01	
...	...	
22	0x51	data
1C	0x52	
...	...	
E3	0xA1	stack
45	0xA2	
...	...	

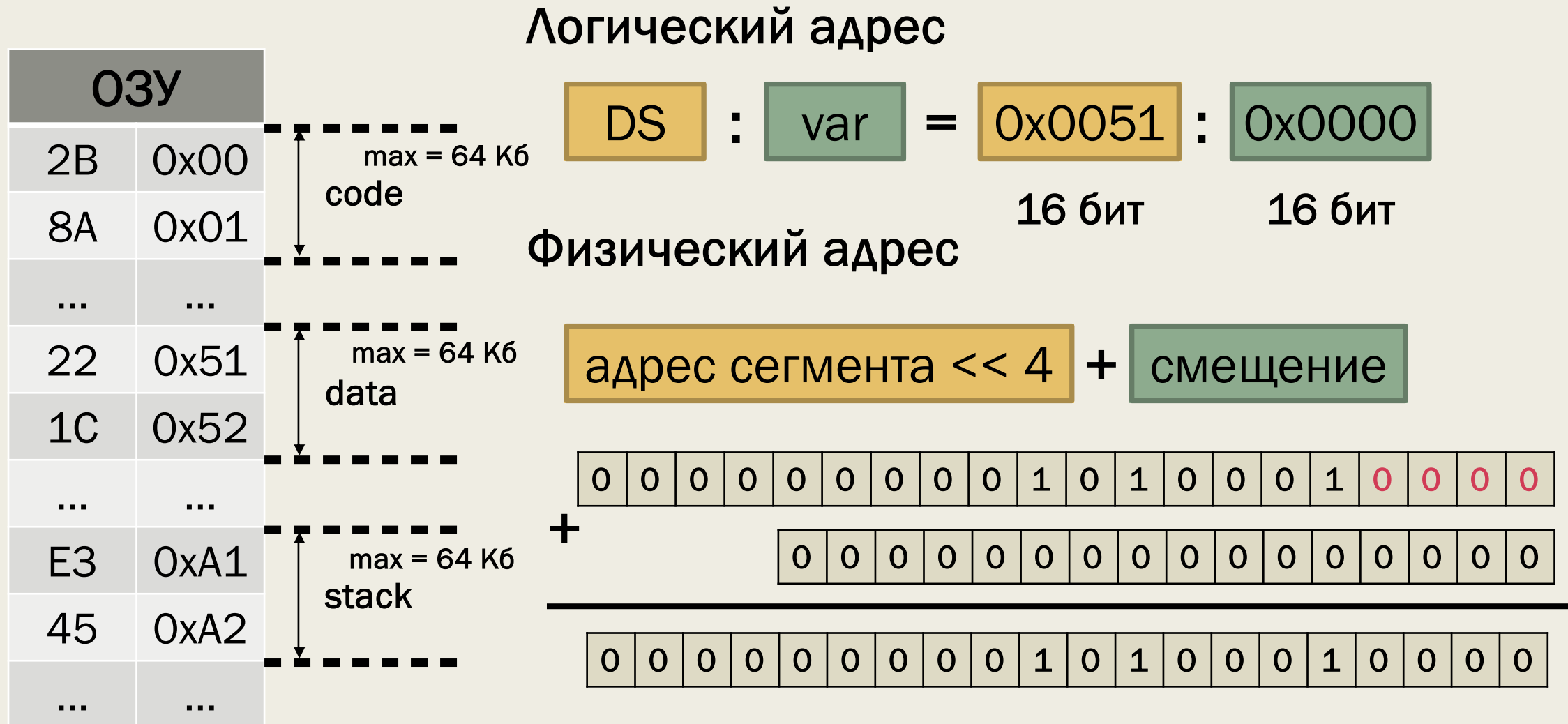
ПРОГРАММА



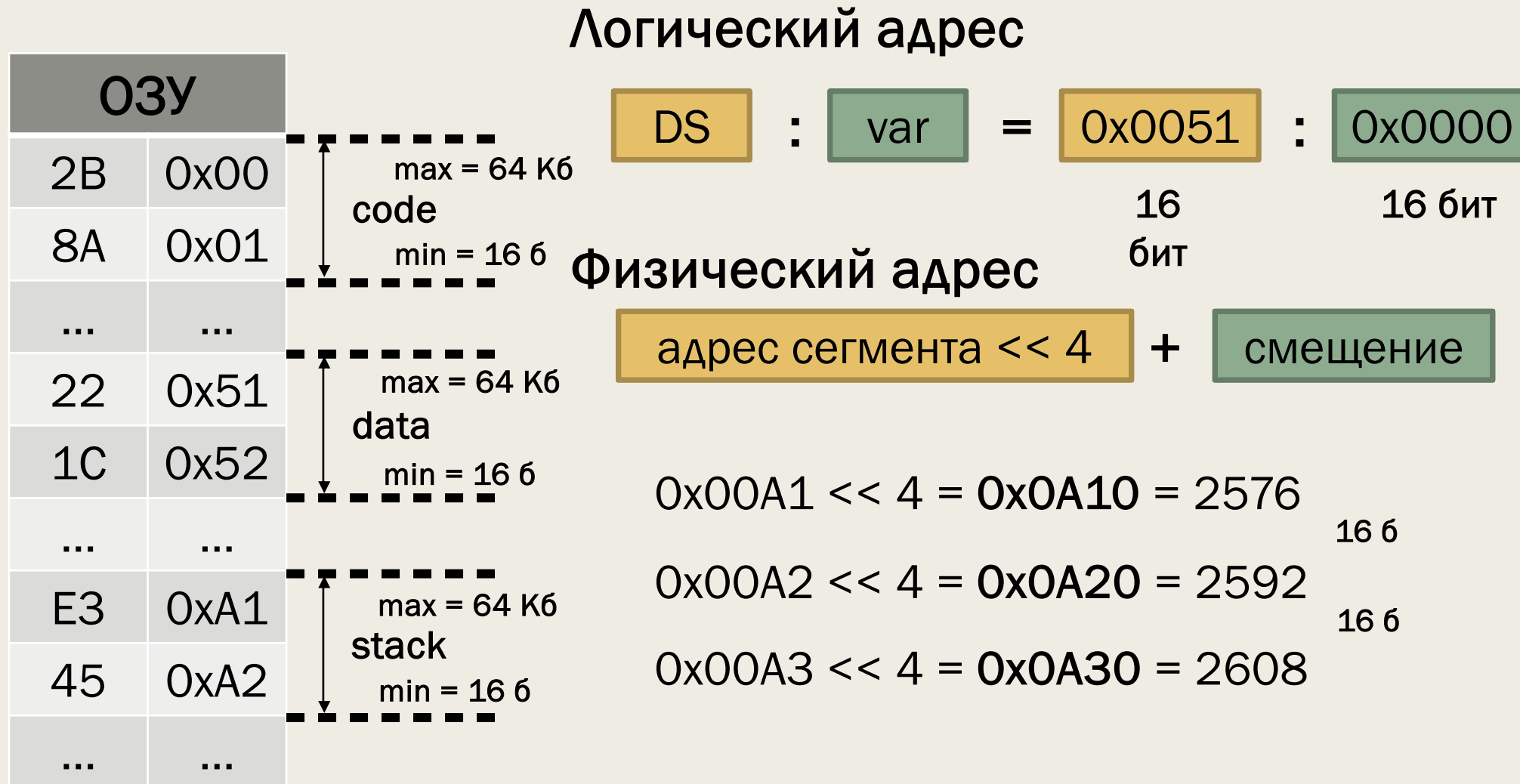
Сегменты программы



Сегменты программы



Сегменты программы



Создание сегментов в коде (1 способ)

```
code SEGMENT разрядность выравнивание класс тип  
.....  
code ends
```

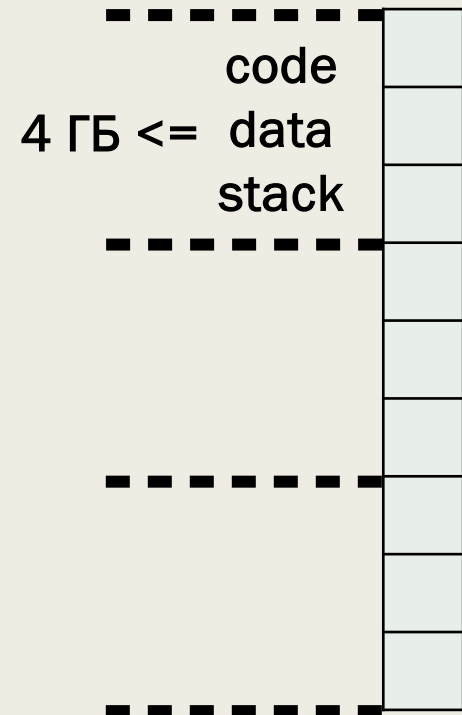
Модели памяти

`.model` (tiny | small | compact | medium | large | huge | flat)

flat – все в одном сегменте, 32-битные сегменты (до 4 ГБ) и адреса

tiny - аналог flat для DOS (сегменты до 64 Кб)

small – используется по одному сегменту памяти на сегмент кода, данных и стека



Создание сегментов в коде (2 способ)

`.model (tiny | small | compact | medium | large | huge | flat)`

`.code` - сегмент кода программы

`.stack [size]` – сегмент стека

`.data` – инициализированные данные

`.data?` – неинициализированные данные

`.const` – неизменяемые данные

`.fardata` – дальние инициализированные данные

`.fardata?` - дальние неинициализированные данные

Компилятор и компоновщик

Компиляция в 2 прохода

Первый проход
Коды операций
...
Директивы
...
Символические имена
...

Первый проход
Объектные модули Windows - .obj, linux - .o
1. Идентификация
2. Таблица точек входа
3. Таблица внешних ссылок
4. Машинные команды и константы
5. Словарь перераспределения
6. Конец модуля

Calling Conventions

Основные соглашения: cdecl, stdcall, fastcall.

Для всех трех соглашений характерно то, что очистка стека выполняется вызываемой подпрограммой.

cdecl (c-declaration) — соглашение о вызовах, используемое компиляторами для языка Си.

Особенности:

- Аргументы функций передаются через стек, справа налево.
- Аргументы, размер которых меньше 4 байт, расширяются до 4 байт.

Calling Conventions

stdcall — соглашение о вызовах, применяемое в ОС Windows для вызова функций WinAPI.

Особенности: аргументы функций передаются через стек, справа налево.

fastcall — общее название соглашений, передающих параметры через регистры.

Особенности: в 32-разрядных компиляторах Microsoft первые два параметра слева направо передаются в регистрах `ecx` и `edx`, остальные параметры – справа налево в стеке.

Ввод-вывод данных в программе на языке Ассемблера

1. Получение дескрипторов.

Используется функция **GetStdHandle**, принимающая на вход 1 параметр.

- 10 – стандартный дескриптор ввода,
- 11 – стандартный дескриптор вывода,
- 12 – стандартный дескриптор сообщений об ошибке.

Функция возвращает дескриптор в регистре EAX.

Ввод-вывод данных в программе на языке Ассемблера

```
EXTERN GetStdHandle@4: PROC
```

```
DIN DD ?; дескриптор ввода; директива DD резервирует память объемом  
; 32 бита (4 байта), знак «?» используется для неинициализированных данных  
DOUT DD ?; дескриптор вывода
```

```
; получим дескриптор ввода
```

```
PUSH -10
```

```
CALL GetStdHandle@4
```

```
MOV DIN, EAX ; переместить результат из регистра EAX
```

```
; в ячейку памяти с именем DIN
```

```
; получим дескриптор вывода
```

```
PUSH -11
```

```
CALL GetStdHandle@4
```

```
MOV DOUT, EAX
```

Ввод-вывод данных в программе на языке Ассемблера

2. Вывод текстовой информации в консоль.

Используется функция **WriteConsoleA**, принимающая на вход 5 параметров:

- 1) дескриптор вывода;
- 2) адрес буфера, в котором находится выводимый текст;
- 3) количество выводимых символов;
- 4) адрес переменной, в которую будет помещено количество действительно выведенных символов;
- 5) резервный параметр, должен быть равен нулю.

Ввод-вывод данных в программе на языке Ассемблера

Таким образом, для вывода информации на консоль нужно совершить следующие действия:

- 1) зарезервировать буфер с текстом (или неинициализированный буфер и ввести текст с клавиатуры);
- 2) получить дескриптор вывода функцией `GetStdHandle`;
- 3) получить длину буфера с текстом.

* Бонусом требуется еще и перекодировать строку

Ввод-вывод данных в программе на языке Ассемблера

Перекодировка строки – функция **CharToOemA**. Принимает 2 аргумента (адрес строки, которую следует перекодировать, и адрес строки, куда следует поместить результат)

```
EXTERN CharToOemA@8: PROC
```

```
STRN DB "Введите строку: ",13,10,0; выводимая строка, в конце добавлены  
; управляющие символы: 13 – возврат каретки, 10 – переход на новую  
; строку, 0 – конец строки; с использованием директивы DB  
; резервируется массив байтов
```

```
MOV EAX, OFFSET STRN; командой MOV значение второго операнда  
; перемещается в первый, OFFSET – операция, возвращающая адрес  
PUSH EAX; параметры функции помещаются в стек командой PUSH  
PUSH EAX  
CALL CharToOemA@8; вызов функции
```

Ввод-вывод данных в программе на языке Ассемблера

Определение длины строки – функция `lstrlenA`. Принимает 1 аргумент (адрес строки)

```
EXTERN lstrlenA@4: PROC; функция определения длины строки
```

```
STRN DB "Введите строку: ",13,10,0; выводимая строка, в конце добавлены  
; управляющие символы: 13 – возврат каретки, 10 – переход на новую  
; строку, 0 – конец строки; с использованием директивы DB  
; резервируется массив байтов
```

```
; определим длину строки STRN  
PUSH OFFSET STRN; в стек помещается адрес строки  
CALL lstrlenA@4; длина в EAX
```


Ввод-вывод данных в программе на языке Ассемблера

```
EXTERN WriteConsoleA@20: PROC
```

```
STRN DB "Введите строку: ",13,10,0; выводимая строка, в конце добавлены  
; управляющие символы: 13 – возврат каретки, 10 – переход на новую  
; строку, 0 – конец строки; с использованием директивы DB  
; резервируется массив байтов
```

```
LENS DD ?; переменная для количества выведенных символов
```

```
; вызов функции WriteConsoleA для вывода строки STRN
```

```
PUSH 0; в стек помещается 5-й параметр
```

```
PUSH OFFSET LENS; 4-й параметр
```

```
PUSH EAX; 3-й параметр
```

```
PUSH OFFSET STRN; 2-й параметр
```

```
PUSH DOUT; 1-й параметр
```

```
CALL WriteConsoleA@20
```

Ввод-вывод данных в программе на языке Ассемблера

3. Вывод числовой и символьной информации в буфер.

Используется функция `wsprintfA`, принимающая на вход переменное число параметров:

- 1) адрес буфера, в который будет помещена строка;
- 2) адрес строки со списком форматов;
- 3) список переменных.

Ввод-вывод данных в программе на языке Ассемблера

```
EXTERN wsprintfA: PROC; т.к. число параметров функции не фиксировано,  
; используется соглашение, согласно которому очищает стек  
; вызывающая процедура
```

```
FMT DB "Число %d", 0; строка со списком форматов для функции wsprintfA
```

```
BUF DB 200 dup (?); буфер для вводимых/выводимых строк длиной 200 байтов
```

```
; вывод числа 123 в буфер BUF
```

```
PUSH 123
```

```
PUSH OFFSET FMT
```

```
PUSH OFFSET BUF
```

```
CALL wsprintfA
```

```
ADD ESP, 12; очистка стека от параметров (изменение регистра ESP  
; на  $3 \cdot 4 = 12$  байтов)
```

```
; вывод строки с числом 123
```

```
PUSH 0
```

```
PUSH OFFSET LENS
```

```
PUSH EAX
```

```
PUSH OFFSET BUF
```

```
PUSH DOUT
```

```
CALL WriteConsoleA@20
```

Ввод-вывод данных в программе на языке Ассемблера

4. Ввод данных с клавиатуры.

Используется функция **ReadConsoleA**, принимающая на вход 5 параметров:

- 1) дескриптор ввода;
- 2) адрес буфера, в который будет помещена вводимая информация;
- 3) длина буфера;
- 4) адрес переменной, в которую будет помещено количество действительно введенных символов;
- 5) резервный параметр, должен быть равен нулю.

Ввод-вывод данных в программе на языке Ассемблера

```
EXTERN ReadConsoleA@20: PROC
```

```
BUF DB 200 dup (?); буфер для вводимых/выводимых строк длиной 200 байтов
```

```
LENS DD ?; переменная для количества выведенных символов
```

```
; ввод строки
```

```
PUSH 0; в стек помещается 5-й параметр
```

```
PUSH OFFSET LENS; 4-й параметр
```

```
PUSH 200; 3-й параметр
```

```
PUSH OFFSET BUF; 2-й параметр
```

```
PUSH DIN; 1-й параметр
```

```
CALL ReadConsoleA@20 ; обратите внимание: LENS больше числа введенных  
; символов на два, дополнительно введенные символы: 13 – возврат каретки и  
; 10 – переход на новую строку
```

Лабораторная работа 1. Особенности

2. По предложенному преподавателем варианту разработать программу на языке Ассемблера, решающую поставленную задачу:

- 1) ввод с клавиатуры 2-х чисел в заданной системе счисления;
- 2) выполнение арифметической операции над этими числами (в предположении, что размер чисел не вызывает переполнения регистров);
- 3) вывод результата в заданной системе счисления.

Все промежуточные данные должны сохраняться в памяти. При выводе результата не использовать функцию `wsprintfA`.

1. Вводите вы строку, а работаете с числом
2. Как работать с отрицательными числами, решаете сами, но работать с ними надо (при этом никто не заставляет использовать дополнительный код)
3. Функцию `wsprintfA` здесь нельзя использовать, чтобы вы научились работать с преобразованием числа в строку