ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ/ 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
- РГЗ (группы ПМ)

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

Структура курса

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
№ п.п.	Наименование дисциплины	Шифр	Объем работы										Распределение по семестрам																
			в зачетных единицах	в часах									Семестр	в зачетку		Экзамены	Зачеты	Виды самостоятельной работы				Число недель теоретического обучения	Установочные лекции	Лекции	Лабор. работы	Практики, семинары	Консультации	Самостоятельная работа	Кафедра, ведущая дисциплину
				Всего	В контактной форме	в т. ч. аудиторная			в том числе, в активных формах	Аттестация	Консультация*	Самостоятельная работа		З.Е.	Часов			Курсовые проекты	Курсовые работы	Расчётно-графические задания (работы), рефераты	Контрольные работы								
						Лекции	Лабор. работы	Практики, семинары																					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
24	Языки программирования и методы трансляции	Б1.В1.24	5	180	60	24	32		18	2	2	120	6	3	108		6					18		12	16		2	78	ПМт ТПИ
													7	2	72		7			7		18		12	16		1	43	
25	Низкоуровневое программирование	Б1.В1.25	3	108	37	12	16		14	2	7	71	5	3	108		5					18		12	16		7	73	ТПИ

Правила аттестации (группы ПМ)

График выполнения и защиты лабораторных работ, РГЗ:
– 1-я лабораторная работа – до 7 недели;
– 2-я лабораторная работа – до 11 недели;
– 3-я лабораторная работа – до 16 недели;
– РГЗ – до 15 недели.

Контрольная неделя 5:
- 0 баллов – студент не начинал показывать лабораторную работу 1.
- 1 балл – студент начал сдавать лабораторную работу 1 (продемонстрировал работу программы, но вынужден ее дорабатывать).
- 2 балла – студент продемонстрировал работу программы и ответил на вопросы преподавателя по программе (!= ответил на контрольные вопросы и полностью защитил работу).

Контрольная неделя 9:
- 0 баллов – лабораторная 1 сдана и защищена, но студент не приступил к сдаче лабораторной работы 2.
- 1 балл – студент начал сдавать лабораторную работу 2 (продемонстрировал работу программы, но вынужден ее дорабатывать).
- 2 балла – студент продемонстрировал работу программы в л.р.2 и ответил на вопросы преподавателя по программе (!= ответил на контрольные вопросы и полностью защитил работу).

Контрольная неделя 13:
- 0 баллов – лабораторные 1-2 сданы и защищены, но студент не приступил к сдаче лабораторной работы 3.
- 1 балл – студент начал сдавать лабораторную работу 3 (продемонстрировал работу программы, но вынужден ее дорабатывать).
- 2 балла – студент продемонстрировал работу программы в л.р.3 и ответил на вопросы преподавателя по программе (!= ответил на контрольные вопросы и полностью защитил работу).

Правила аттестации (группы ПМИ)

График выполнения и защиты лабораторных работ:
– 1-я лабораторная работа – до 7 недели;
– 2-я или 3-я лабораторная работа – до 11 недели;
– 3-я лабораторная работа – до 16 недели.

Контрольная неделя 5:
- 0 баллов – студент не начинал показывать лабораторную работу 1.
- 1 балл – студент начал сдавать лабораторную работу 1 (продемонстрировал работу программы, но вынужден ее дорабатывать).
- 2 балла – студент продемонстрировал работу программы и ответил на вопросы преподавателя по программе (!= ответил на контрольные вопросы и полностью защитил работу).

Контрольная неделя 9:
- 0 баллов – лабораторная 1 сдана и защищена, но студент не приступил к сдаче лабораторной работы 2.
- 1 балл – студент начал сдавать лабораторную работу 2(3) (продемонстрировал работу программы, но вынужден ее дорабатывать).
- 2 балла – студент продемонстрировал работу программы в л.р.2(3) и ответил на вопросы преподавателя по программе (!= ответил на контрольные вопросы и полностью защитил работу).

Контрольная неделя 13:
- 0 баллов – лабораторные 1-2(3) сданы и защищены, но студент не приступил к сдаче лабораторной работы 4.
- 1 балл – студент начал сдавать лабораторную работу 4 (продемонстрировал работу программы, но вынужден ее дорабатывать).
- 2 балла – студент продемонстрировал работу программы в л.р.4 и ответил на вопросы преподавателя по программе (!= ответил на контрольные вопросы и полностью защитил работу).

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

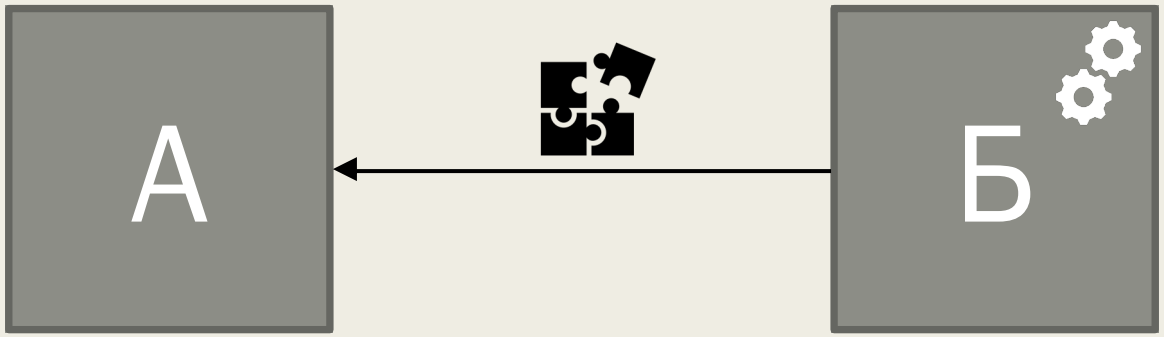
Семейство (архитектура) процессоров

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

Семейство процессоров

8086	80186	80286	80386	80486	…	CORE i7	…

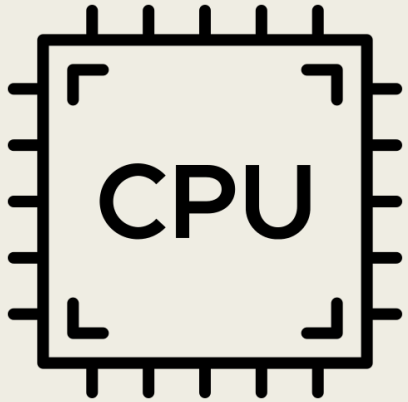
Обратная совместимость – все программы, написанные для модели А, должны работать для модели Б.



Семейство (архитектура) процессоров

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**

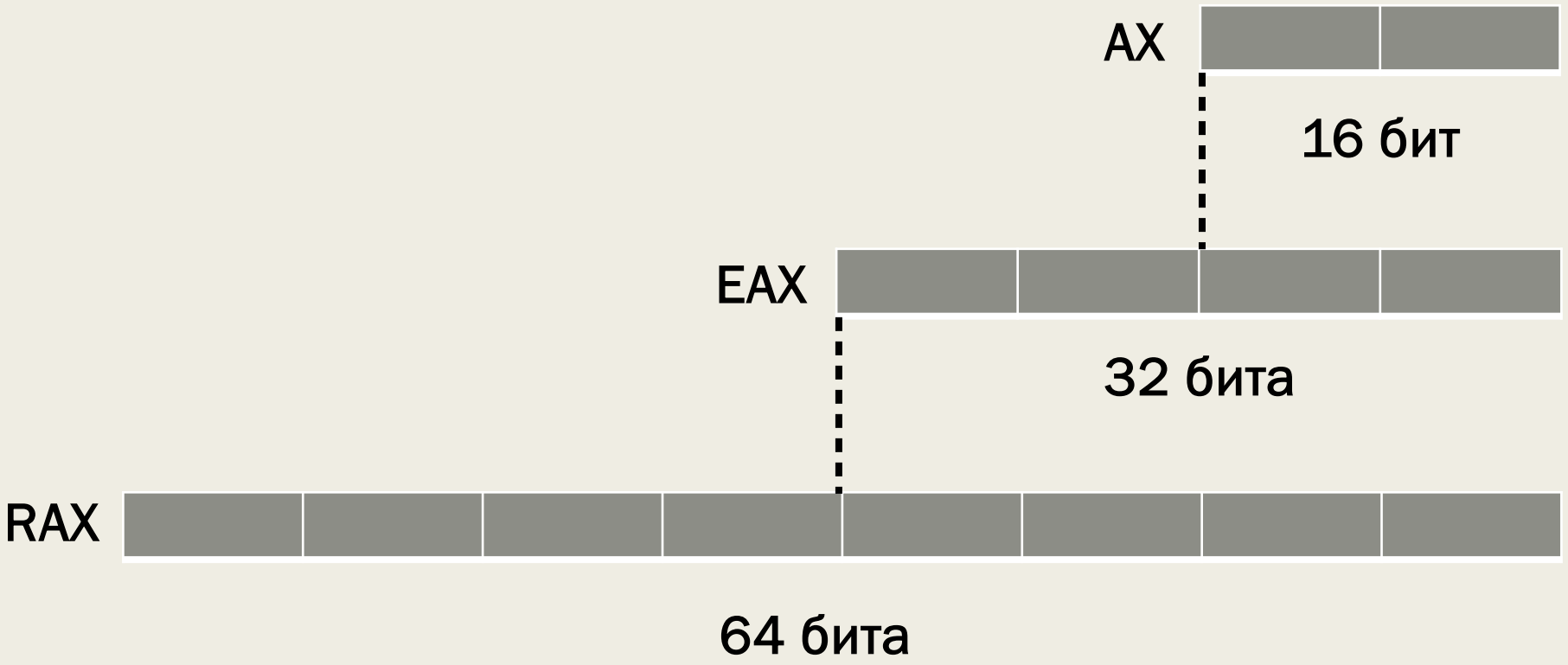
Регистры-указатели: **SP BP IP**

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

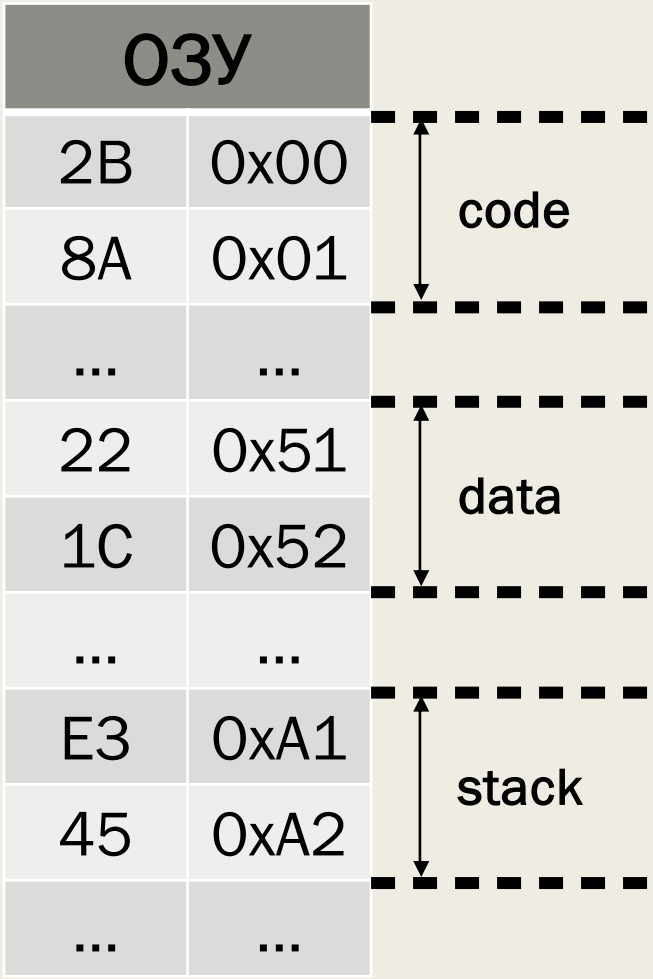
```
movw %ax, %bx
```

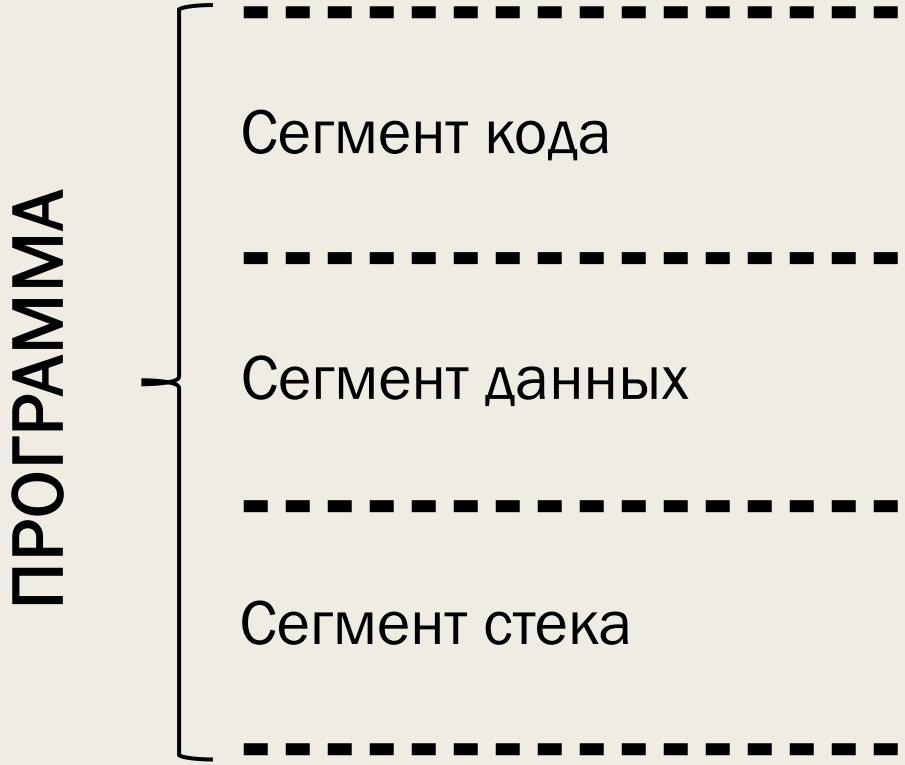
b (byte) – операнды размером в 1 байт

w (word) – операнды размером в 1 слово

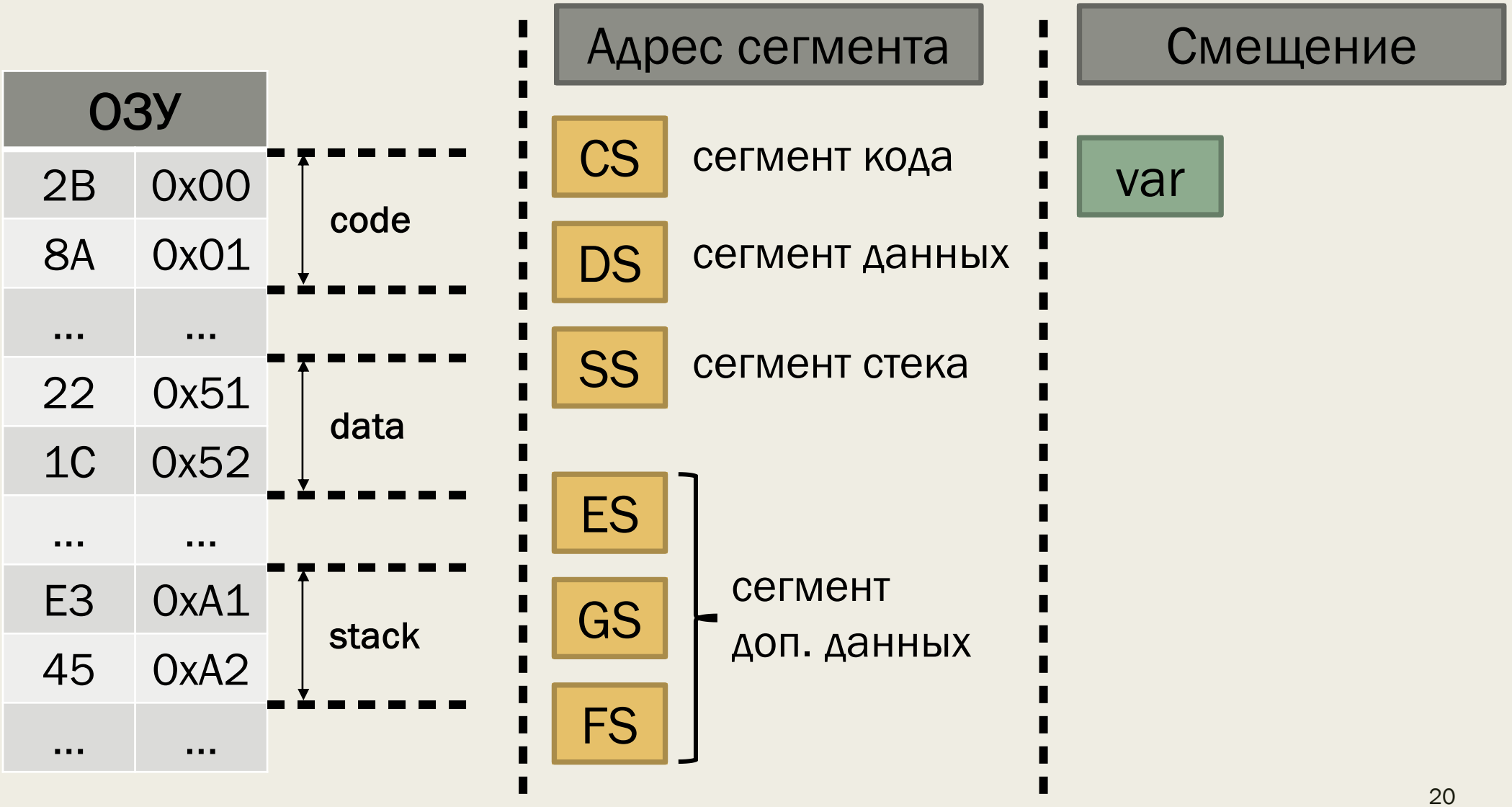
l (long) – операнды размером в 4 байта

Сегменты программы

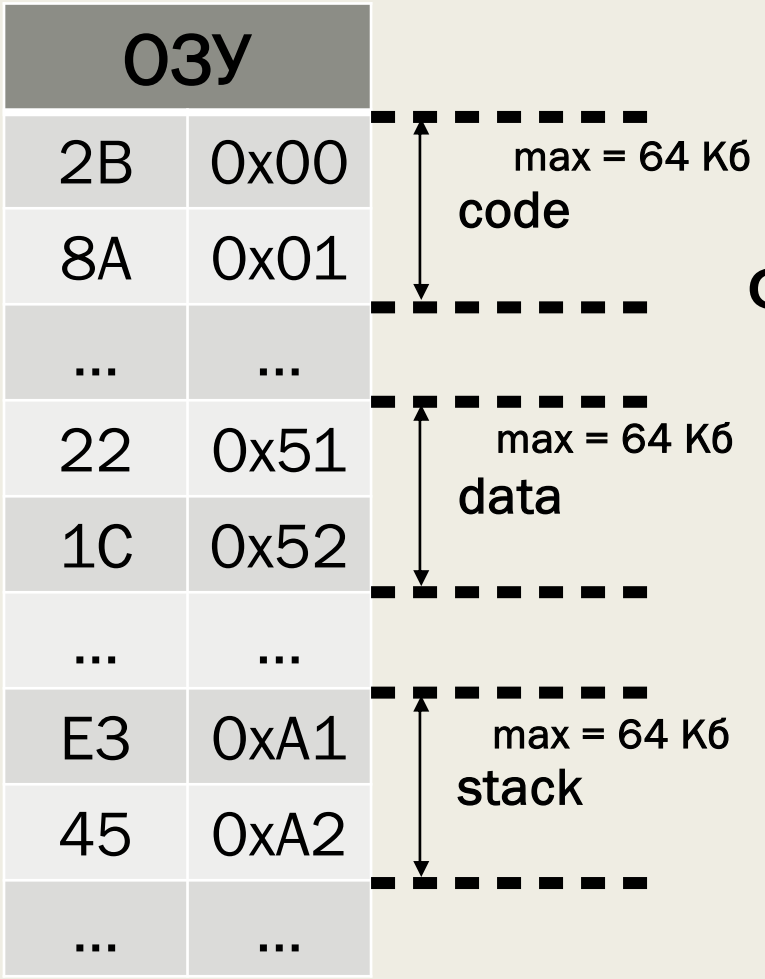




Сегменты программы



Сегменты программы



Логический адрес

DS : var = 0x0051 : 0x0000

16 бит 16 бит

Физический адрес

адрес сегмента << 4 + смещение

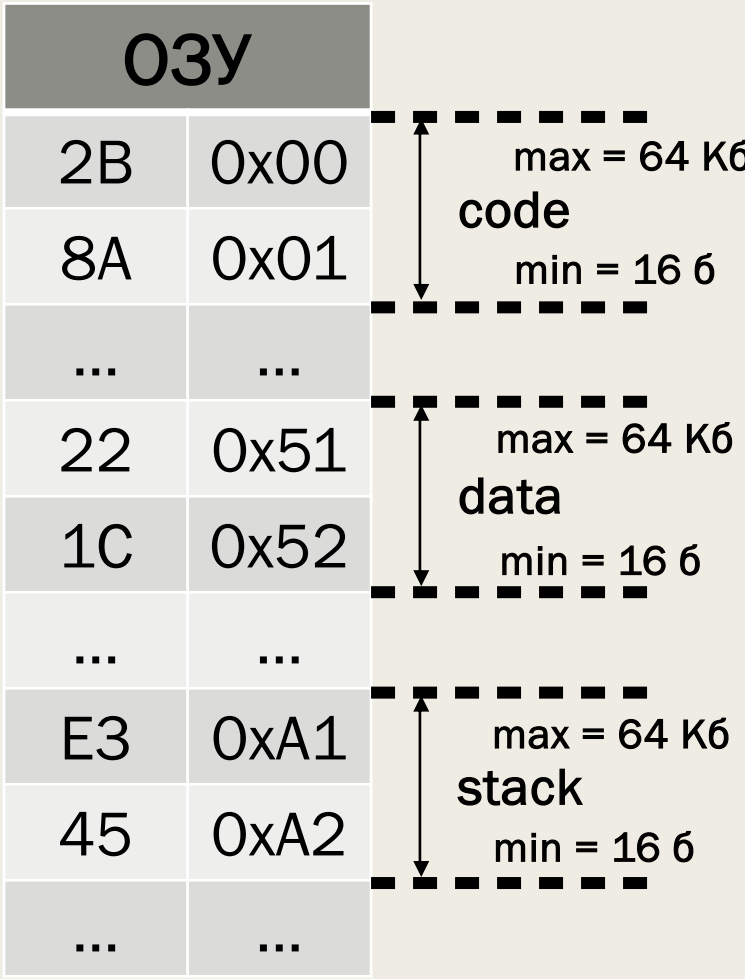
0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0

\+

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0

Сегменты программы



Логический адрес

DS : var = 0x0051 : 0x0000

16 бит — 16 бит

Физический адрес

адрес сегмента << 4 + смещение

0x00A1 << 4 = **0x0A10** = 2576

0x00A2 << 4 = **0x0A20** = 2592

0x00A3 << 4 = **0x0A30** = 2608

16 б

16 б

Создание сегментов в коде (1 способ)

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

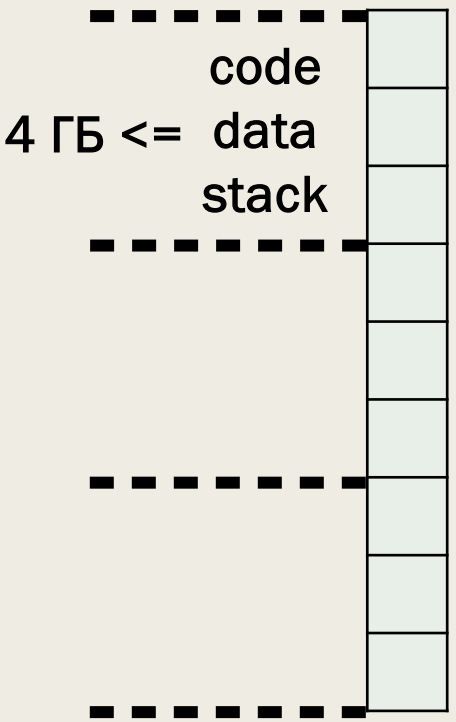
Модели памяти

.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
…

Второй 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*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 числа в заданной системе счисления (система счисления А);

2) вывод введенного числа в десятичной системе счисления;

3) вычисление значения полинома вида $ax^2 + bx + c$ (в предположении, что результат вычислений не приводит к переполнению регистров);

4) вывод результата в заданной системе счисления (система счисления Б);

5) вывод результата в десятичной системе счисления.

Все промежуточные данные должны сохраняться в памяти. При выводе результата не использовать функцию wsprintfA. **Приглашение к вводу числа, а также вывод результатов на экран должны сопровождаться понятными пользователю текстовыми сообщениями.**

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

Лабораторная работа 2-3. Особенности

Лабораторная работа 2: по предложенному преподавателем варианту разработать программу на языке Ассемблера, решающую поставленную задачу **с использованием цепочечных команд**

Лабораторная работа 3: все то же самое, что в л.р. 2, + соглашения о вызовах (**использовать цепочечные команды - обязательно!**)

Лабораторная работа 4 (РГЗ). Особенности

2. Написать программу, реализующую вычисление функции в точке (вводится пользователем) в соответствии с заданным вариантом.
Программа должна состоять из модулей на C++ и ассемблере, причем в модуле на C++ осуществляется ввод-вывод, а все вычисления – в модуле на ассемблере.

Не забываем о том, что регистры сопроцессора нужно инициализировать.

Первый фрагмент программы с командами сопроцессора должен начинаться с команды инициализации сопроцессора **FINIT**, приводящей сопроцессор в некоторое начальное состояние.