**Лекция 8. Обратная разработка. Основы исполнения программ и ассемблера.**

Данная лекция рассчитана на появление базовых знаний об исполнение программ, их работе и получениям основ языка ассемблера.

Для понимания того, как выполняется программа – необходимо понимать архитектуру современных компьютеров. Современные компьютеры строятся по принципам архитектуры фон Неймана:

1) Принцип однородности памяти.

Данный принцип говорит о том, что в памяти одновременно хранятся как данные программы, так и команды (код программы). Был и другой вариант данного принципа в Гарвардской архитектуре. В ней, данные и инструкции хранились отдельно друг от друга.

2) Принцип адресности.

Память программы можно представлять, как последовательность ячеек, пронумерованных целыми числами и каждую ячейку можно прочитать из памяти или записать в неё значение, просто обратившись к ней по её номеру.

3) Принцип программного управления

Выполнение программы управляется посредством последовательности команд, эти команды выполняются одна за другой до тех пор, пока не будет выполнена специальная команда, вроде условного перехода, безусловного перехода или вызова функции.

4) Принцип двоичного кодирования.

Вся информация (данные и команды) кодируются двоичными числами.

Коротко о **сегментации памяти**:

Оперативная память, используемая в программах, написанных на С/С++ разделена на области двух типов:

* Сегменты данных
* Сегменты кода (текстовые сегменты)

В сегментах кода содержится код программы. Обычно данные сегменты защищаются от записи, то есть ОС следит за тем, чтобы данные находящиеся в этих сегментах не изменялись.

В сегментах данных располагаются данные программы (значения переменных, массивы и пр.).

При запуске программы выделится два сегмента данных:

* Сегмент глобальных данных
* Стек (для локальных переменных)

В процессе работы программы могут выделяться и освобождаться дополнительные сегменты памяти.

Обращение к адресу вне выделенных сегментов (к сегментам, которые не принадлежат вашей программе) – ошибка времени выполнения (access violation, segmentation fault).

Коротко рассмотрим вопрос **выполнения программы.**

В скомпилированном коде вашей программы каждой функции будет соответствовать отдельная секция. Адрес начала этой секции – это адрес функции, тот адрес, который будет подставлен в место вызова вашей функции.

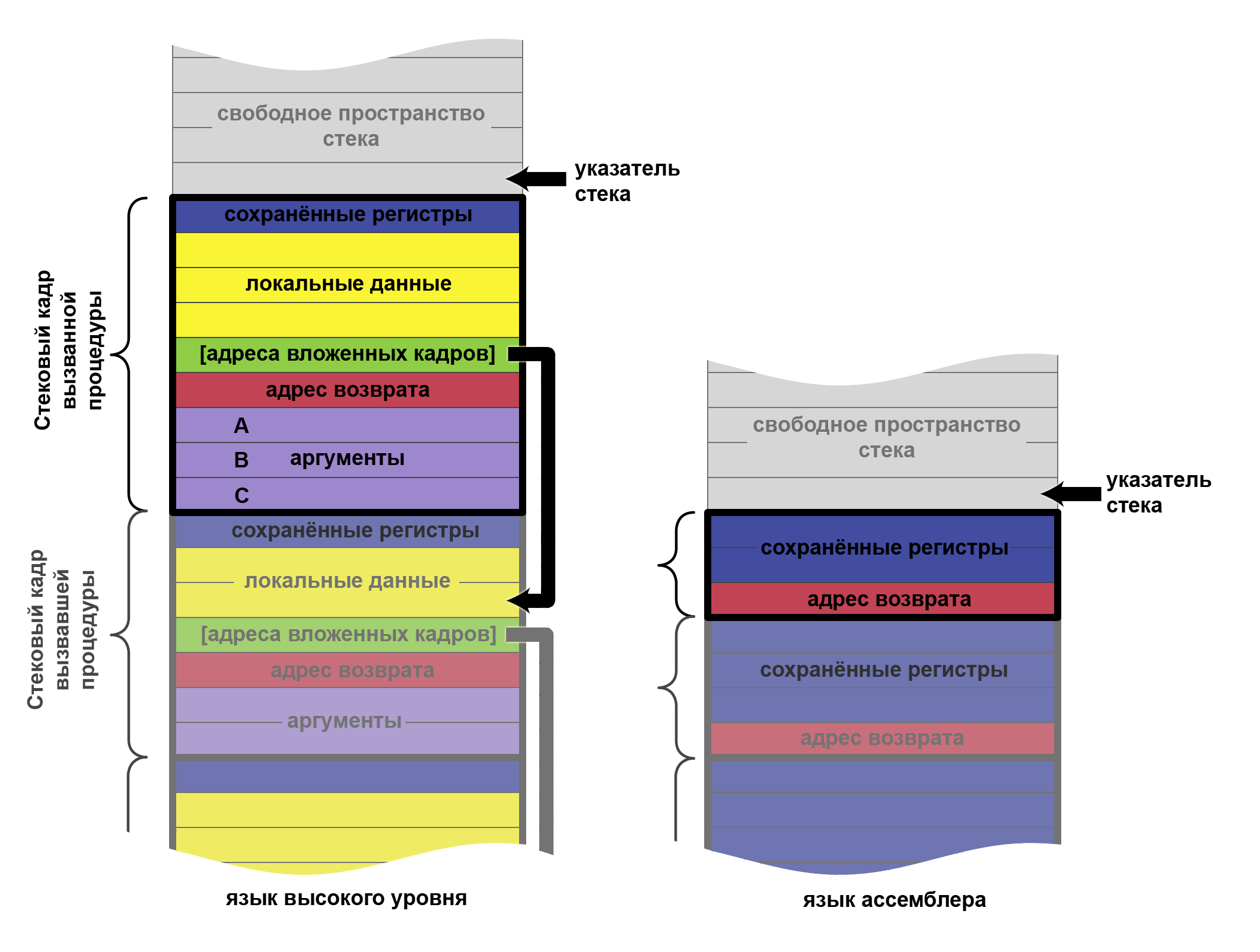
Телу функции соответствует последовательность команд процессора, которая получилась после компиляции. Работа с данными, такими как локальные или глобальные переменные происходит на уровне байт, то есть никакой информации о типах нет.

В процессе выполнения адрес каждой следующей инструкции хранится в специально регистре процессора IP (Instruction pointer), который указывает на адрес следующей инструкции. При выполнении инструкций этот регистр увеличивается и таким образом инструкции выполняются последовательно. В тот момент, когда встречается какая-либо специальная инструкция (условный/безусловный переход, вызов функции) IP изменяется и выполнение переходи на другую инструкцию, которая может быть в другом месте программы. Например, при вызове функции IP переводится на адрес начала функции.

Рассмотрим **стек вызовов.**

Стек вызовов – это сегмент данных, используемый для хранения локальных переменных, временных значений, адресов возврата, аргументов, передаваемых в функции. Стек выделяется при запуске программы и обычно он небольшой по размеру (примерно 4 Мб).

На стеке хранятся локальные переменные функции, которая исполняется в данный момент. При выходе из функции, соответствующая область стека, где находились локальные переменные данной функции объявляется свободной, и другая функция может её перезаписать.



Коротко об **соглашении о вызове.**

**Соглашение о вызове** — описание технических особенностей вызова подпрограмм, определяющее:

* способы передачи параметров подпрограммам;
* способы вызова (передачи управления) подпрограмм;
* способы передачи результатов вычислений, выполненных подпрограммами, в точку вызова;
* способы возврата (передачи управления) из подпрограмм в точку вызова.

Соглашение о вызове описывает следующее:

* способ передачи аргументов в функцию. Варианты:
  + аргументы передаются через регистры процессора;
  + аргументы передаются через стек;
  + смешанные (соответственно, стандартизируется алгоритм, определяющий, что передаётся через регистры, а что — через стек или другую память):
  + первые несколько аргументов передаются через регистры; остальные — через стек (небольшие аргументы) или другую память (большие аргументы);
  + аргументы небольшого размера передаются через стек, большие аргументы — через другую память;
* порядок размещения аргументов в регистрах и/или стеке. Варианты:
  + слева направо или прямой порядок: аргументы размещаются в том же порядке, в котором они перечислены при вызове функции. Достоинство: машинный код соответствует коду на языке высокого уровня;
  + справа налево или обратный порядок: аргументы передаются в порядке от конца к началу. Достоинство: упрощается реализация функций, принимающих произвольное число аргументов (например, printf()) (так как на вершине стека оказывается всегда первый аргумент);
* код, ответственный за очистку стека:
  + код, вызывающий функцию, или вызывающая функция. Достоинство: возможность передачи в функцию произвольного числа аргументов;
  + код самой функции или вызываемая функция. Достоинство: уменьшение количества инструкций, необходимых для вызова функции (инструкция для очистки стека записывается в конце кода функции и только один раз);
* конкретные инструкции, используемые для вызова и возврата. Для процессора x86, работающего в защищённом режиме, используются исключительно инструкции call и ret; при работе в стандартном режиме используются инструкции call near, call far и pushf/call far (для возврата соответственно retn, retf и iret);
* способ передачи в функцию указателя на текущий объект (this или self) в объектно-ориентированных языках. Варианты (для процессора x86, работающего в защищённом режиме):
  + как первый аргумент;
  + через регистр ecx или rcx;
* код, ответственный за сохранение и восстановление содержимого регистров до и после вызова функции:
  + вызывающая функция;
  + вызываемая функция;
* список регистров, подлежащих сохранению/восстановлению до/после вызова функции.

Соглашение о вызове может быть описано в документации к ABI архитектуры, в документации к ОС или в документации к компилятору.

**Основы ассемблера**.

Язык ассемблера — система обозначений, используемая для представления в удобочитаемой форме программ, записанных в машинном коде. Язык ассемблера позволяет программисту пользоваться алфавитными мнемоническими кодами операций, по своему усмотрению присваивать символические имена регистрам ЭВМ и памяти, а также задавать удобные для себя схемы адресации (например, индексную или косвенную). Кроме того, он позволяет использовать различные системы счисления (например, десятичную или шестнадцатеричную) для представления числовых констант и даёт возможность помечать строки программы метками с символическими именами с тем, чтобы к ним можно было обращаться (по именам, а не по адресам) из других частей программы (например, для передачи управления)

**Набор команд:**

Синтаксис языка ассемблера определяется системой команд конкретного процессора.

Типичными командами языка ассемблера являются (большинство примеров даны для Intel-синтаксиса архитектуры x86):

* Команды пересылки данных (mov и др.)
* Арифметические команды (add, sub, imul и др.)
* Логические и побитовые операции (or, and, xor, shr и др.)
* Команды управления ходом выполнения программы (jmp, loop, ret и др.)
* Команды вызова прерываний (иногда относят к командам управления): int
* Команды ввода-вывода в порты (in, out)

**Некоторые команды:**

mov eax, 123 ; поместить значение 123 в регистр eax

add eax, 321 ; прибавить к значению в регистре eax число 321

sub eax, 321 ; отнять от значения в регистре eax число 321

jmp loc\_deadbeef ; безусловный переход на именованную метку

push eax ; положить значение регистра eax на стек

pop eax ; получить значение с вершины стека и поместить в регистр eax

call strcmp ; вызвать функцию strcmp

ret ; возврат управления в вызывающую функцию (адрес берётся с вершины стека)