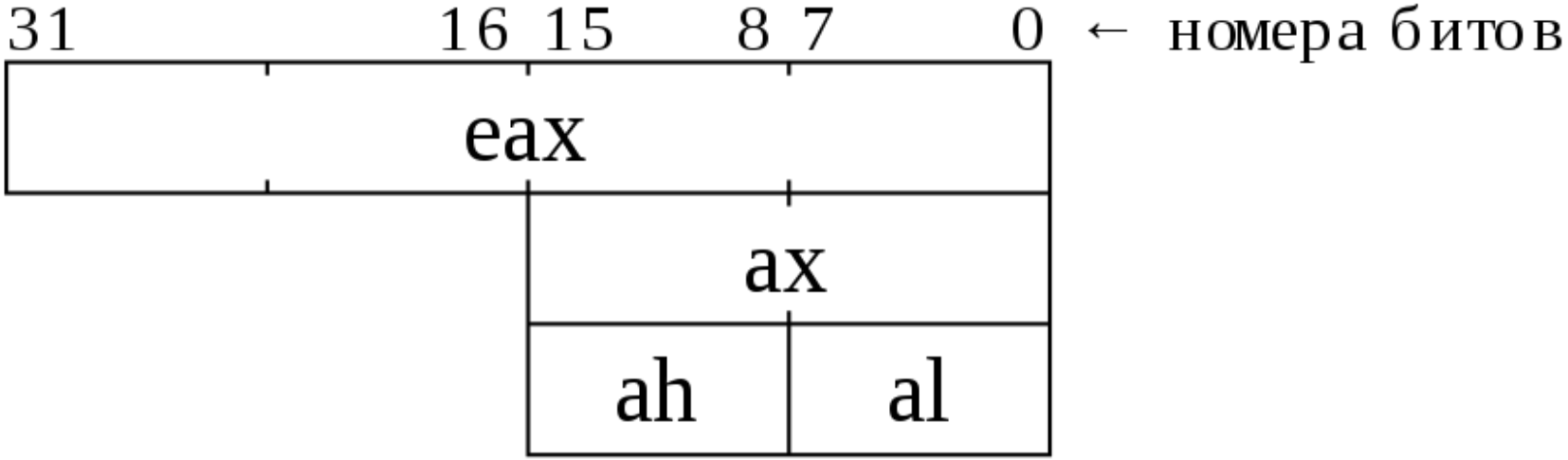
Семинар 3. Введение в ассемблер

Размерность регистра



Секции кода в ассемблере

- Данные помещаются в секцию данных - директива **.data**
- Команды (и, возможно, константные данные) помещаются в секцию кода - директива **.text**
- Неинициализированные данные (зарезервированные) определяются специальной секции - директива **.bss**
- Константные данные можно поместить в отдельную секцию (часть секции кода) - директива **.section .rodata**

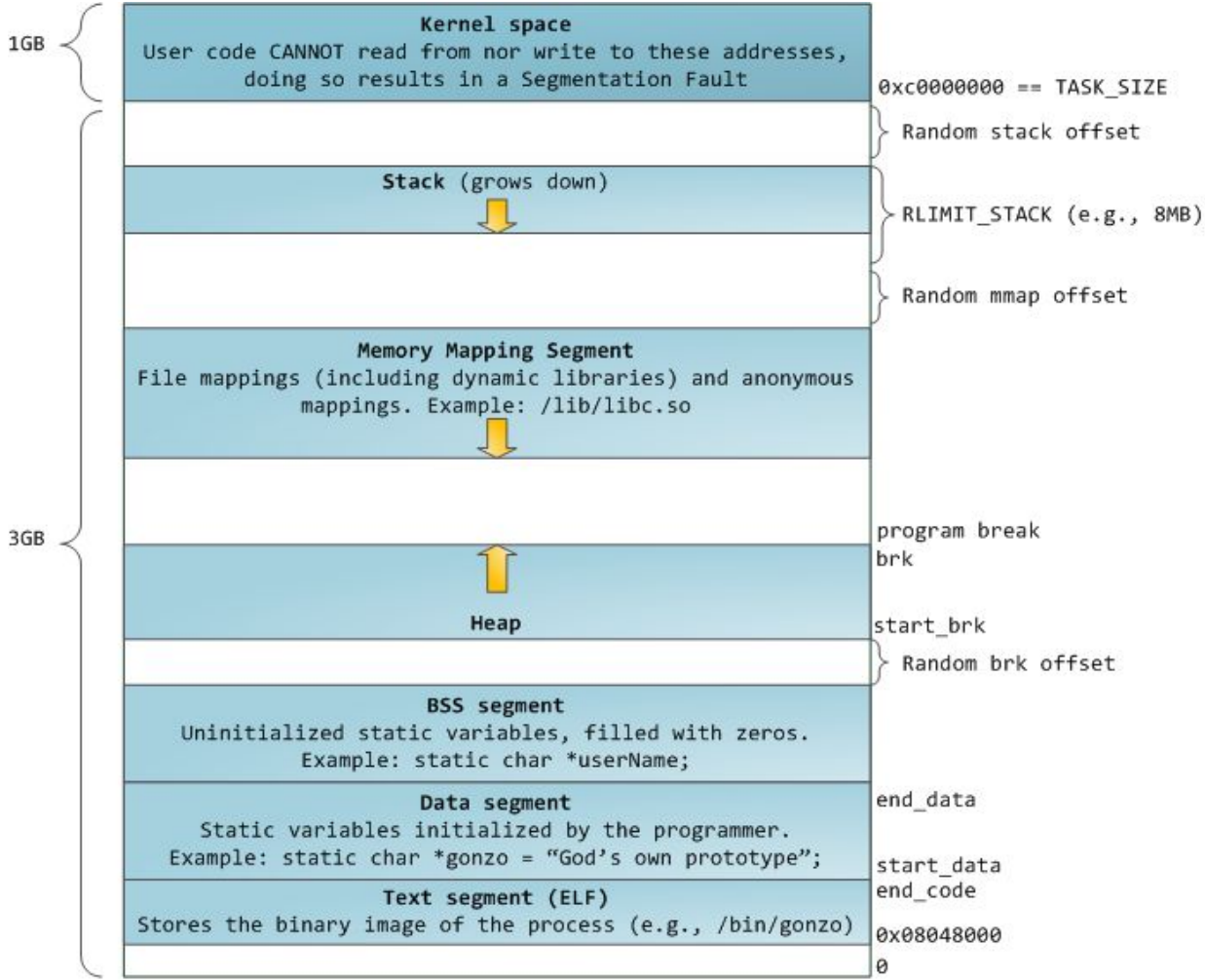
Данные

- **.ascii "STR"** размещает строку STR. Нулевых байтов не добавляет
- **.string "STR"** размещает строку STR, после которой следует нулевой байт (как в языке C)
- У директивы **.string** есть синоним **.asciz** (z от англ. zero — ноль, указывает на добавление нулевого байта)
- Строка-аргумент этих директив может содержать стандартные escape-последовательности

Неинициализированные данные

- В скомпилированной программе секция **.bss** не занимает места
- **.space количество_байт**

Сегменты памяти



Работа с памятью

- Синтаксис: **смещение(база, индекс, множитель)**
- Вычисленный адрес будет равен **база + индекс × множитель + смещение**
- База и Индекс - регистры процессора
- Множитель может принимать значения 1, 2, 4 или 8

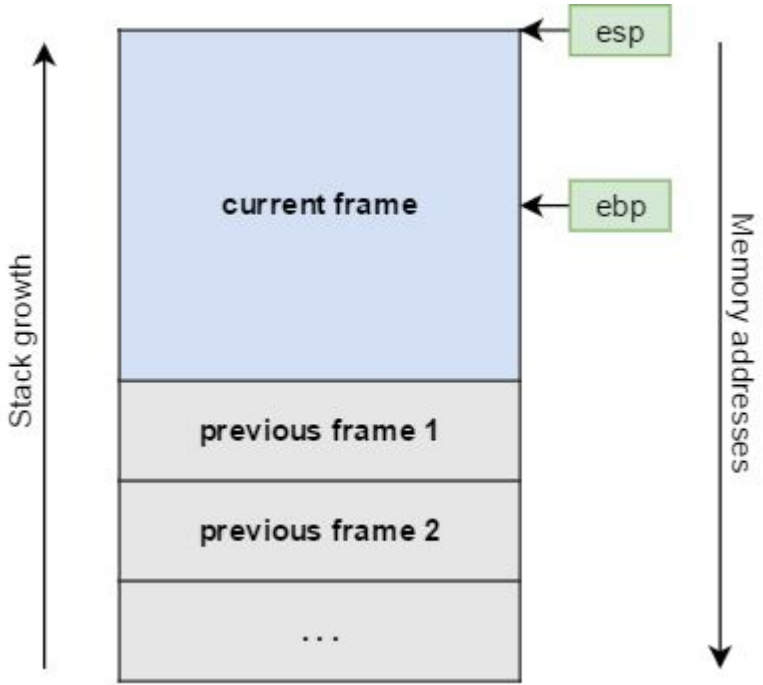
Работа с памятью

- **(%eax)** - адрес находится в регистре **%eax**
- **4(%ecx)** - операнд находится по адресу **%ecx + 4** (удобно адресовать поля структур)
- **foo(,%ecx,4)** - операнд находится по адресу **foo + %ecx × 4** (удобно итерироваться по массиву)

Стек вызовов

- Стек является неотъемлемой частью архитектуры процессора и поддерживается на аппаратном уровне: в процессоре есть специальные регистры и команды для работы со стеком
- Обычно стек используется для сохранения адресов возврата и передачи аргументов при вызове процедур, также в нём выделяется память для локальных переменных
- Стек располагается в оперативной памяти
- Стек растёт в сторону младших адресов

Стек вызовов



Работа со стеком

- При операции **push АРГУМЕНТ** значение **%esp** уменьшается на размер элемента в байтах, новый элемент записывается по адресу, на который указывает **%esp**
- При операции **pop АРГУМЕНТ** содержимое памяти по адресу, который записан в **%esp**, записывается в регистр, а значение адреса в **%esp** увеличивается на размер элемента в байтах

Подпрограммы

- **call МЕТКА** - вызов подпрограммы
- **ret [NUM]** - извлечь из стека новое значение регистра **%eip** [и если команде передан операнд **NUM**, **%esp** увеличивается на это число - убираются аргументы из стека]
- Подпрограмма может изменить значения регистров, но обязана сохранить **%ebp, %ebx %esi %edi, %esp**. Сохранение остальных регистров - задача программиста
- Возвращаемое значение - через регистр **%eax**

Передача аргументов подпрограмме

- Через регистры (и, если их много, через стек)
- Через стек - кладем аргументы на стек (в прямом или обратном порядке - соглашение) и вызываем подпрограмму

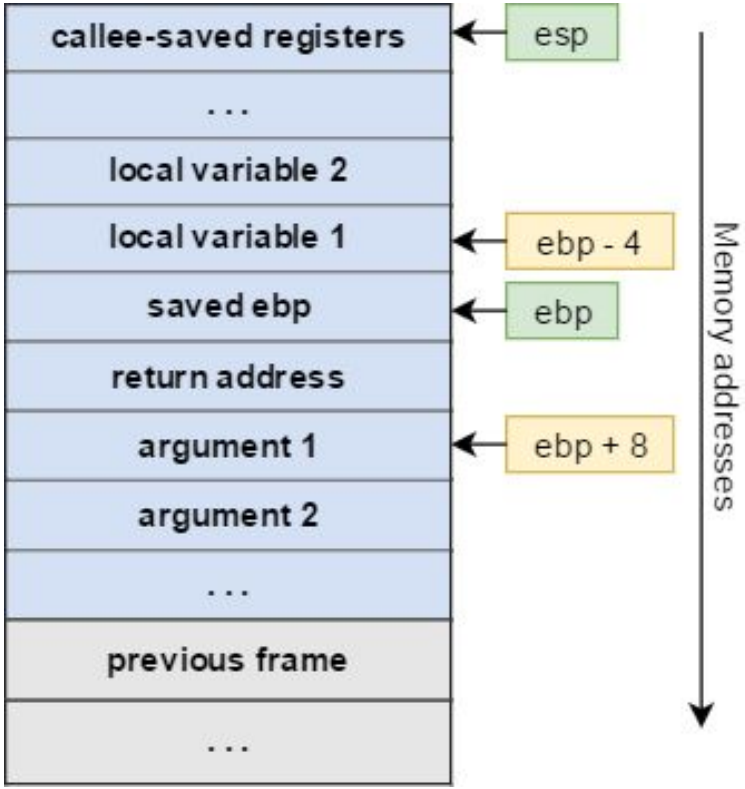
Правила вызывающей стороны

- Сохраняем регистры, которые не обязаны быть сохранены вызываемой стороной (если в них содержатся наши данные)
- Записываем аргументы на стек в обратном порядке (например, сначала значение, потом форматную строку для printf)
- Вызываем подпрограмму
- Удаляем аргументы функции (добавляем к стеку значение в байтах)
- Восстанавливаем регистры из стека

Правила вызываемой стороны

- Сохраняем значение **%ebp** на стеке
- Устанавливаем значение **%esp** в **%ebp** (новый кадр стека, можем легко обращаться к переданным аргументам)
- Выделяем место на стеке для локальных переменных (уменьшить значение **%esp**)
- Сохраняем в стек регистры, которые мы обязаны сохранить (если их не меняем - можно не сохранять)

Правила вызываемой стороны



Правила вызываемой стороны

- Восстанавливаем сохраненные регистры
- Освобождаем место под локальные переменные (просто установка значения **%ebp** в **%esp**)
- Восстанавливаем **%ebp**
- Возвращаемся при помощи **ret**