

Лекция 8. Работа стека при вызове функции по соглашениям ABI

План лекции

Процесс вызова функции

Работает вызывающая функция F

Работает вызванная функция f_1

Кадр стека (`stack frame`)

Возврат из вызванной функции в вызывающую

Работает вызванная функция f_1

Работает вызывающая функция F

Выводы

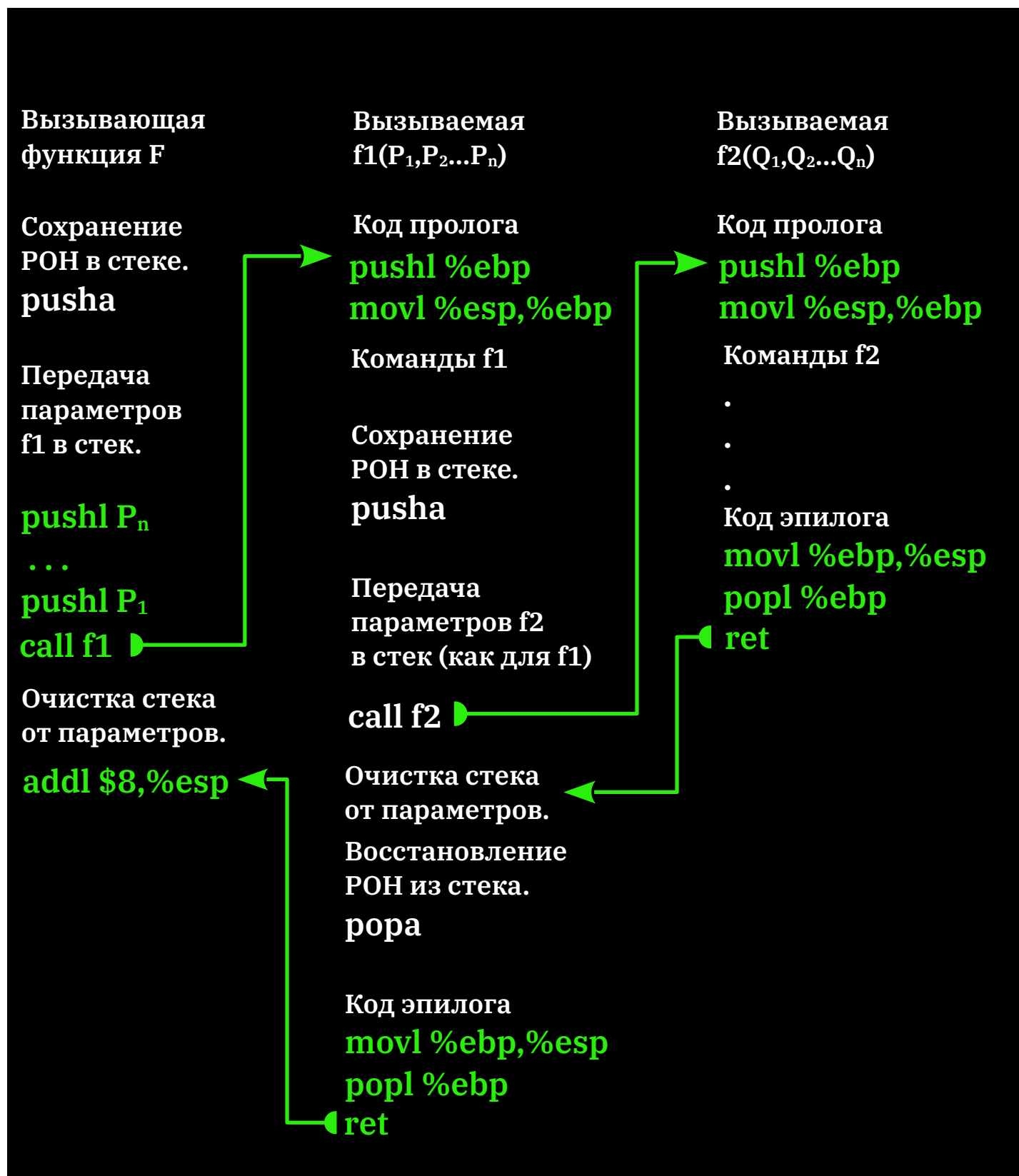


Рис. 1. Схема организации функций по соглашениям о вызовах ABI

Опишем последовательно выполнение команд согласно общей схеме на Рис 1. Сразу после запуска вызывающей функции стек находится в исходном состоянии:

Исходное состояние	В стеке	Операнд для доступа
Вершина стека	1	0(%esp)

Процесс вызова функции

Работает вызывающая функция

0. Выполняется команда `pusha` сохранения в стеке регистров общего назначения `%eax`, `%ecx`, `%edx`, `%ebx`, `%esp`, `%ebp`, `%esi`, `%edi`.

NB. В стек записывается значение `%esp`, которое он имел ДО выполнения команды `pusha`. Стек переходит в состояние 0:

Состояние 0	В стеке	Операнд для доступа
Вершина стека	<code>%edi</code>	0(%esp)
	<code>%esi</code>	4(%esp)
	<code>%ebp</code>	8(%esp)
	<code>%esp</code>	12(%esp)
	<code>%ebx</code>	16(%esp)
	<code>%edx</code>	20(%esp)
	<code>%ecx</code>	24(%esp)
	<code>%eax</code>	28(%esp)
Старший адрес	1	32(%esp)

1. Выполняются команды записи параметров в стек в порядке обратном определенному в функции $f(P1, P2... Pn)$ языка C:

```
pushl Pn
pushl Pn-1
```

...
pushl P1

После их выполнения стек переходит в состояние 1:

Состояние 1	В стеке	Операнд для доступа
Вершина стека	P1	0(%esp)
	P2	4(%esp)
	...	...
	PN	4*(N-1)(%esp)
	Восемь РОН функции F	Не показаны.
Старший адрес	1	Не показан.

Здесь и далее *не приводятся* формы операндов элемента стека в режиме регистровой адресации для РОН.

NBNB. Если параметр не 4-байтовое слово, а некоторая структура данных, например массив, то в стек нужно помещать 4-байтовый адрес этой структуры, например первого элемента массива.

2. Выполняется команда вызова функции

<адрес команды call> call f1

<адрес след. команды> след. команда

Команда call производит два действия:

- Запись в стек значения <адрес след. команды>, т. н. адреса возврата, который будем обозначать RetAddr, что эквивалентно выполнению команды:
 - pushl RetAddr
- Передача управления на f1 — <%eip> = <адрес f1>

Стек переходит в состояние 2:

Состояние 2	В стеке	Операнд для доступа
Вершина стека	RetAddr функции F	$0(\%esp)$
	P1	$4(\%esp)$
	P2	$8(\%esp)$
	...	...
	PN	$4+4*(N-1)(\%esp)$
Старший адрес	Восемь РОН функции F	Не показаны.
	1	Не показан.

Работает вызванная функция f1

1. Выполняются ДВЕ ОБЯЗАТЕЛЬНЫЕ команды пролога

```
pushl %ebp    # записать в стек значение бывшее в вызывающей
movl %esp,%ebp # сделать %ebp - базовым для доступа к данным,
               # созданным в стеке при вызове f1
```

Поясним назначение и работу этих команд.

1. `pushl %ebp` # записать в стек значение бывшее в вызывающей

Эта команда включена в пролог по двум причинам.

1. Соглашения ABI предусматривают произвольную глубину

вложенности вызовов функций (т. е. в теле f1 можно вызвать еще функцию, в ее теле еще и т.д). При таких вызовах опять будет использовать регистр %esp.

2. В теле `f1` целесообразно иметь многократный адресный доступ к значениям параметров и, возможно, к другим данным в стеке, с помощью базового регистра, которым, из-за первой причины NBNB не может быть регистр `%esp`, т. к. его значение изменится при вложенном вызове и его нельзя будет использовать как значение базового регистра.

В ABI для многократного адресного доступа к данным, созданным в стеке при вызове функции, требуется использовать в качестве базового регистра `%ebp`. Поэтому значение, которое он имел в вызывающей функции:

- запоминается в стеке при выполнении кода пролога;
- и будет восстановлено при возврате в вызывающую функцию при выполнении кода эпилога.

Таким образом это значение также входит в состав данных, создаваемых в стеке при вызове `f1`.

II. `movl %esp, %ebp # сделать %ebp - базовым для доступа к данным
в стеке, созданным при вызове f1`

Эта команда выполняется NBNB ДО любых вложенных в `f1` вызовов функций пока значение `%esp` НЕ ИЗМЕНЕНО и, следовательно, может быть использовано как базовое для доступа к данным, созданным в стеке при вызове `f1`. Далее регистр `%ebp`, а не `%esp` следует использовать в командах тела функции как базовый для адресного доступа к данным, созданным в стеке при вызове `f1` и к локальным переменным в стеке же (см. чуть ниже), если программист их создаст.

NBNB. Т.о. значение регистра `%ebp` в теле вызываемой функции изменять нельзя!

Будем обозначать значение `%ebp` которое было в вызывающей функции в момент передачи управления в `f1` как `caller%ebp` функции `F`.

После выполнения команды `pushl %ebp` стек переходит в состояние 3:

Состояние 3	В стеке	Операнд для доступа
Вершина стека	<code>caller%ebp</code> функции <code>F</code>	<code>0(%esp)</code>
	<code>RetAddr</code> функции <code>F</code>	<code>4(%esp)</code>
	<code>P1</code>	<code>8(%esp)</code>
	<code>P2</code>	<code>12(%esp)</code>
	<code>...</code>	<code>...</code>
	<code>PN</code>	<code>4+4+4*(N-1)(%esp)</code>
	Восемь <code>POH</code> функции <code>F</code>	Не показаны.
Старший адрес	<code>1</code>	Не показан.

NB. ЗНАЧЕНИЕ `%esp` ПОКА НЕЛЬЗЯ МЕНЯТЬ (например для вызова функции из `f1`), В Т.Ч. КОМАНДАМИ `pushl` и `popl` Т.К. ПОТЕРЯЕМ БАЗОВОЕ ЗНАЧЕНИЕ ДЛЯ ДОСТУПА К ПАРАМЕТРАМ.

После выполнения команды `movl %esp, %ebp` стек переходит в состояние 4:

Состояние 4	В стеке	Операнд для доступа
Вершина стека	caller%ebp функции F	0(%esp), 0(%ebp)
	RetAddr функции F	4(%esp), 4(%ebp)
	P1	8(%esp), 8(%ebp)
	P2	12(%esp), 12(%ebp)
	...	...
	PN	$4+4+4 \cdot (N-1)(\%esp)$ $4+4+4 \cdot (N-1)(\%ebp)$
	Восемь PОН функции F	Не показаны.
Старший адрес	1	Не показан.

NBNB. Теперь, после выполнения второй команды пролога оба регистра — %esp и %ebp можно использовать как базовые для адресного доступа к элементам стека, прежде всего, к параметрам. Однако регистр %esp нужен нам для других целей, прежде всего для возможного вызова других функций из текущей функции — f1, для создания переменных, локальных в f1, возможно для чего-то еще.

Главное — теперь МОЖНО использовать %esp для любых целей т. к. в качестве базового регистра для адресного доступа к параметрам стандартно по соглашениям ABI используется регистр %ebp.

Также теперь ясна еще одна цель первой команды пролога pushl %ebp, сохраняющей в стеке значение %ebp вызывающей функции — ведь в ней он **NBNB** также используется как базовый для доступа к ее параметрам в ее стеке, сформированном ранее при ее

вызове. Заметим еще раз, что при выходе из функции значение `%ebp`, которое было в вызвавшей эту функцию, восстанавливается командами кода эпилога. Эти команды рассмотрим далее.

Итак, после выполнения кода пролога регистр `%esp` можно использовать для любых целей. Обычно это:

- резервирование в стеке памяти для переменных, локальных в `f1`;
- вызов другой функции (на Рис.1 — `f2`) из кода текущей функции `f1`.

Память для локальных переменных `f1` в стеке резервируется соответствующим уменьшением значения `%esp`. Отметим, что после выполнения кода эпилога значения этих переменных становятся недоступными.

Память для двух локальные 4-х байтовых переменных резервируется выполнением после кода пролога команды:
`subl $8, %esp` после чего стек переходит в состояние 5:

Состояние 5	В стеке	Операнд для доступа
	Начало кадра стека (stack frame) <code>f1</code>	
Вершина стека	Локальная .2 функции <code>f1</code>	<code>-8(%ebp)</code>
	Локальная .1 функции <code>f1</code>	<code>-4(%ebp)</code>
	<code>caller%ebp</code> функции <code>F</code>	<code>0(%ebp)</code>
	<code>RetAddr</code> функции <code>F</code>	<code>4(%ebp)</code>
	<code>P1</code>	<code>8(%ebp)</code>
	<code>P2</code>	<code>12(%ebp)</code>

Старший адрес	...	...
	PN	$4+4+4 * (N-1) (\%ebp)$
	Восемь PОН функции F	Не показаны.
	Конец кадра стека (stack frame) f1	
	1	Не показан.

NBNB. В этом состоянии ОБЕСПЕЧЕН многократный адресный доступ к параметрам и локальным переменным f1 в стеке через базовый регистр %ebp как к элементам одномерного массива. При этом регистр %esp уже не нужен для работы f1 и может использоваться для любых целей.

Кадра стека (stack frame)

Структура, создаваемая в стеке по соглашениям ABI при вызове функции называется принадлежащим ей кадром стека времени выполнения (a frame on the run-time stack), далее просто кадр стека (stack frame). Он обозначен строками в предыдущей таблице из которой также ясен состав данных кадра стека. При этом говорят, что регистр %ebp является базой кадра стека.

Примеры адресного доступа к параметрам и локальным переменным

Запись значения первого параметра в регистр %edx:

```
movl 8(%ebp), %edx
```

Вычисление суммы значений второго параметра и регистра %edx:

```
addl 12(%ebp), %edx
```

Запись содержимое регистра %eax в локальную переменную 1:

```
movl %eax, -4(%ebp)
```

Вычисление суммы значений локальной переменной 2 и регистра %edx:

```
addl -8(%ebp), %edx
```

NBNB. Память в стеке для локальных переменных БУДЕТ НЕДОСТУПНА после выхода из функции.

NBNB. Локальные переменные не будут удалены при записи параметров в стек при вложенном вызове другой функции из f1, их значения могут быть использованы в f1 после возврата из этой другой функции.

Отметим, что при вложенном вызове вызываемая функция будет формировать свой кадр стека, который будет расположен в стеке «выше» (в направлении младших адресов ОП) кадра стека вызывающей функции.

Таким образом глубина вложенности вызовов функций ограничивается объемом памяти выделенным архитектурному стеку. В таблице ниже показано состояние стека после вызова функции f2 и выполнения команд ее пролога. Кадры стека f1 и f2 выделены рамками разных цветов.

Состояние стека после выполнения кода пролога f2	В стеке	Операнд для доступа
	Начало кадра стека (stack frame) f2	
Вершина стека	Локальная . 2 функции f2	- 8 (%ebp)
	Локальная . 1 функции f2	- 4 (%ebp)
	caller%ebp функции f1	0 (%ebp)
	RetAddr функции f1	4 (%ebp)
	Q1	8 (%ebp)
	Q2	12 (%ebp)
	. . .	. . .
	Qm	$4 + 4 + 4 * (N - 1) (%ebp)$
	Восемь P0H функции f1	Не показаны.
Конец кадра стека (stack frame) f2		

Старший адрес	Начало кадра стека (stack frame) f1	
	Локальная .2 функции f1	- 8 (%ebp)
	Локальная .1 функции f1	- 4 (%ebp)
	caller%ebp функции F	0 (%ebp)
	RetAddr функции F	4 (%ebp)
	P1	8 (%ebp)
	P2	12 (%ebp)
	...	...
	PN	$4 + 4 + 4 * (N - 1) (\%ebp)$
	Восемь PОН функции F	Не показаны.
	Конец кадра стека (stack frame) f1	
	1	Не показан.

Возврат из вызванной функции в вызывающую

Проницательный студент уже подозревает, что код эпилога выполняет действия, обратные коду пролога, убирая из стека кадр вызванной функции, вместе с командами очистки стека от параметров в вызывающей функции. Рассмотрим процесс возврата из функции f1.

Работает вызванная функция f1

1. Выполняется первая команда эпилога:

```
movl %ebp, %esp
```

и `%esp` получает значение, которое он имел до выполнения команды

```
subl $8, %esp
```

выделившей память для локальных переменных, значения которых теперь становятся недоступными, а стек возвращается из состояния 5 к состоянию 4:

Состояние 4	В стеке	Операнд для доступа
Вершина стека	caller%ebp функции F	0(%esp), 0(%ebp)
	RetAddr функции F	4(%esp), 4(%ebp)
	P1	8(%esp), 8(%ebp)
	P2	12(%esp), 12(%ebp)
	...	...
	PN	$4+4+4^*(N-1)(\%esp)$ $4+4+4^*(N-1)(\%ebp)$
	Восемь PОН функции F	Не показаны.
Старший адрес	1	Не показан.

2. Выполняется вторая команда эпилога:

```
popl %ebp
```

которая подготавливает к работе вызывающую функцию, восстанавливая (и удаляя из стека) значение `%ebp`, которое он имел до выполнения команды `call`.

Стек переходит из состояния 4 в состояние 2:

Состояние 2	В стеке	Операнд для доступа
Вершина стека	RetAddr функции F	$0(\%esp)$
	P1	$4(\%esp)$
	P2	$8(\%esp)$
	...	...
	PN	$4+4*(N-1)(\%esp)$
Старший адрес	Восемь PОН функции F	Не показаны.
	1	Не показан.

NBNB. Сейчас на вершине стека находится адрес возврата в вызывающую функцию. Именно это и нужно для выполнения команды `ret`. Она читает (и удаляет) из стека значение `RetAddr` и, помещая его в регистр `%eip`, передает управление вызывающей функции. При этом стек переходит в состояние 1:

NBNB. В стеке сейчас только параметры и PОН.

Работает вызывающая функция

Выполняется команда очистки стека от параметров:

`addl $L,%esp`

(`L` — константа, равная суммарной длине в байтах всех параметров — для 2 параметров это 8). Стек переходит в состояние 0:

Состояние 0	В стеке	Операнд для доступа
Вершина стека	%edi	0(%esp)
	%esi	4(%esp)
	%ebp	8(%esp)
	%esp	12(%esp)
	%ebx	16(%esp)
	%edx	20(%esp)
	%ecx	24(%esp)
	%eax	28(%esp)
Старший адрес	1	32(%esp)

Выполняется команда восстановления РОН:

ра и стек переходит в исходное состояние:

Исходное состояние	В стеке	Операнд для доступа
Вершина стека	1	0(%esp)

Выводы

- 1) По соглашениям о связях ABI команды передачи параметров в стек, команда `call`, команды пролога и, по желанию программиста, команды резервирования памяти для локальных переменных, создают в стеке для каждой вызываемой функции отдельную, принадлежащую ей структуру данных — кадр стека.
- 2) Операнды команд для адресного доступа к параметрам функции и локальным переменным в кадре стека задаются в виде смещение(%ebp), где смещение выбирается из таблицы

состояния стека 5.

В частности, смещения параметров положительные — первого 8 — форма операнда 8 (ebp), второго 12 — форма операнда 12 (ebp) и т. д. Смещения локальных переменных в стеке отрицательные — первой - 4 — форма операнда - 4 (ebp), следующей - 8 — форма операнда - 8 (ebp) и т. д.

- 3) После формирования кадра стека регистр %esp больше не нужен для работы вызванной функции, что обеспечивает возможность вложенных вызовов других функций с формированием их собственных кадров стека.
- 4) В каждый момент времени существует столько кадров стека, сколько функций не выполнили код эпилога и команду ret.
- 5) Код эпилога обеспечивает выход из функции и деактивацию ее кадра стека, которая завершается в вызывающей функции командами очистки стека от параметров.