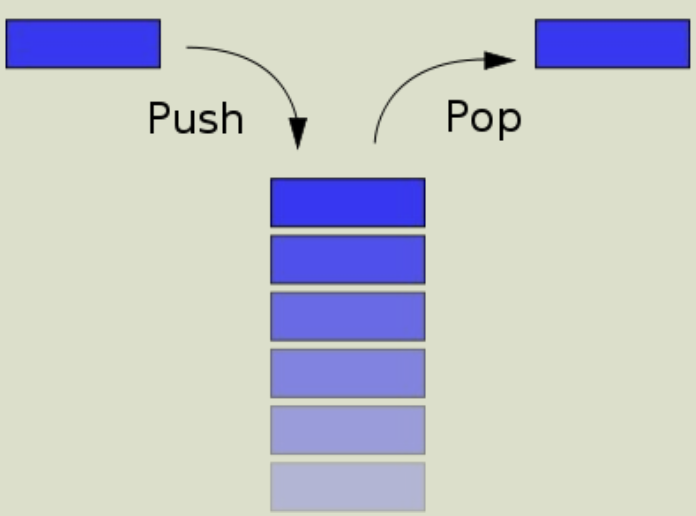
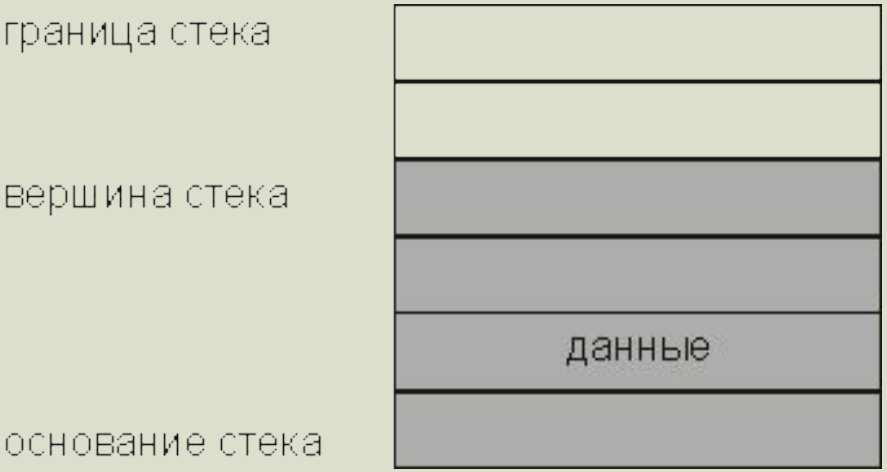
Стек и организация подпрограмм

Стек

Стек – это определенный динамический способ хранения данных, при котором в каждый момент времени доступ возможен только к одному из элементов, а именно к тому, который был занесен в стек последним.

Стек — структура данных с методом доступа к элементам LIFO (англ. Last In — First Out, «последним пришел — первым вышел»).

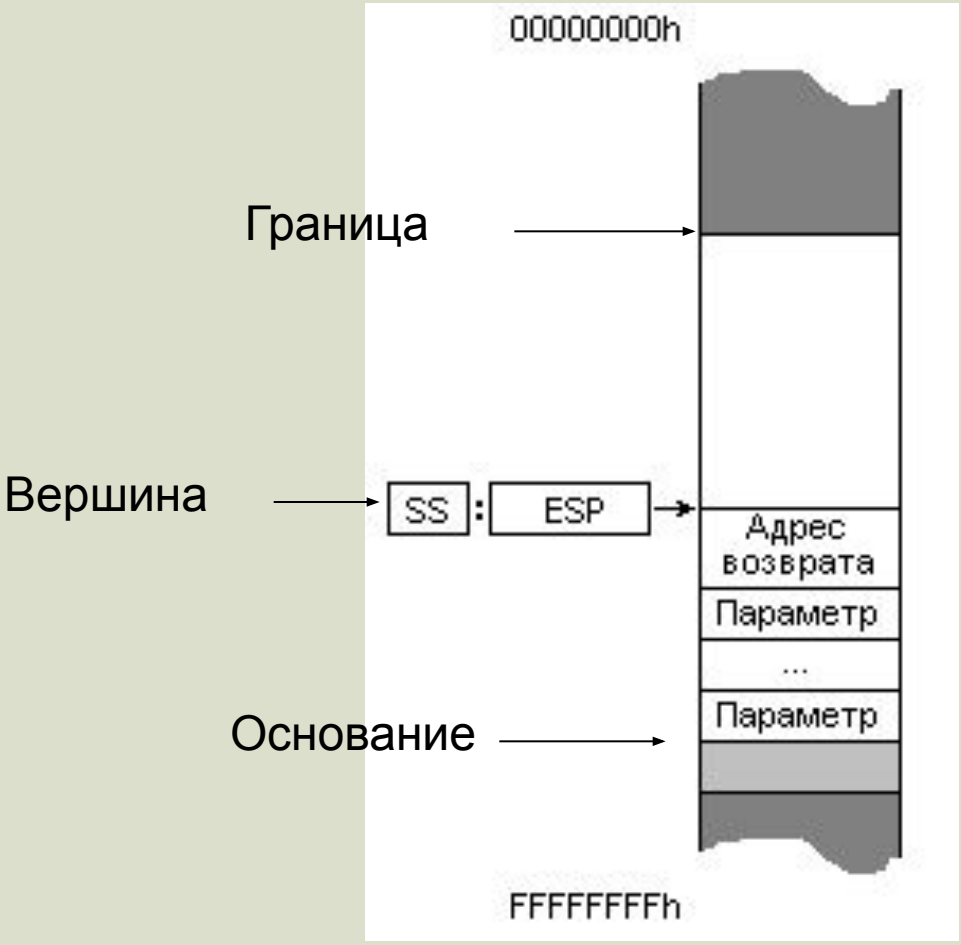




Стек и организация подпрограмм

Реализация стека на уровне ЦП

Регистры x86:
SS – хранит границу стека (начало сегмента стека)
ESP – хранит вершину стека (адрес элемента, записанного в стек последним)
EBP – регистр для манипуляций со стеком. Позволяет получить доступ к предпоследнему, предпредпоследнему, и т.д. элементу.



Стек и организация подпрограмм

Реализация стека на уровне ЦП

Команды для работы со стеком:

PUSH – добавить данные в стек. Команды записывает данные в стек и уменьшает регистр ESP.

POP – извлечь данные из стека. Команда извлекает данные и увеличивает ESP.

```
PUSH BX
PUSH CX
<здесь любые манипуляции с BX и CX>
POP CX
POP BX
```

Стек и организация подпрограмм

Реализация стека на уровне ЦП

Регистры x86:
SS – хранит границу стека (начало сегмента стека)
ESP – хранит вершину стека (адрес элемента, записанного в стек последним)
EBP – регистр для манипуляций со стеком. Позволяет получить доступ к предпоследнему, предпредпоследнему, и т.д. элементу.

```
mov EBP,ESP          ;получаем доступ к последнему
inc [EBP]            ;можем изменить последний
add EBP,2            ;получаем доступ к предпоследнему
inc [EBP]            ;можем изменить предпоследний
```

Стек и организация подпрограмм

Организация подпрограмм

Подпрограмма — поименованная или иным образом идентифицированная часть программы, содержащая описание определённого набора действий.

Преимущества подпрограмм:

1. Возможность повторного использования программного кода.
2. Структуризация программы с целью удобства её понимания и сопровождения.

Особенности использования функций:

1. Функции могут вызывать друг друга.
2. Как правило заранее нельзя предсказать, какие функции будут вызваны, и в какой последовательности.
3. Возможен рекурсивный вызов функций, причем необязательно напрямую: 1-я функция может вызвать 2-ю, а та, в свою очередь – 1-ю.

Стек и организация подпрограмм

Организация подпрограмм

```
function func1 (Param:integer): integer
var x,y;
begin
      if … then
      begin
            func1(x);
      end;
…
end;
function func2 (Param:real): integer
var z,y;
begin
      func1(y);
      func2(z);
…
end;
```

Стек и организация подпрограмм

Проблемы:

Организация подпрограмм

Каждый раз при вызове функции необходимо хранить адрес возврата. Таким образом, может оказаться, что в некоторый момент времени необходимо хранить множество таких адресов.
func1() □ func2() □ func3() □ func2()…
После окончания работы функции такой адрес необходим, чтобы вернуться в предыдущую функцию.

Для каждой вызванной функции необходимо отдельно хранить ее локальные переменные и передаваемые параметры. Это связано с тем, что возможен рекурсивный вызов функции, т.е. в один момент времени могут выполняться несколько экземпляров функции, каждый со своими параметрами.

Стек и организация подпрограмм

Организация подпрограмм

Решение:
Адреса возврата, передаваемые параметры и локальные переменные хранятся в стеке.

Таким образом, последними загружаются в стек значения последней вызванной функции, а при их извлечении из стека, следующими будут значения предыдущей, и т.д.

Обобщенный алгоритм вызова функции:
- Поместить в стек параметры, передаваемые функции
- Поместить в стек адрес возврата (адрес следующая команды, после вызова функции).
- Функция работает. При необходимости, локальные переменные также размещаются в стеке.
- Извлечь из стека все значения и перейти по адресу возврата, который находится в стеке.

Стек и организация подпрограмм

Поддержка на уровне ЦП: Команды x86:

Организация подпрограмм

CALL <имя подпрограммы(метка)>
Помещает адрес возврата в стек, передает управление подпрограмме

RET
Извлекает из стека адрес и передает управление по этому адресу.

Команда **call** заносит в стек адрес следующей команды

```
…
mov ax,2
mov bx,3
call proc1          ;ax=5
…
proc1 PROC
   add ax,bx
   ret
proc1 ENDP
```

Команда **ret** извлекает из стека адрес и инициирует переход по нему

Стек и организация подпрограмм

Пример с передачей параметров

Организация подпрограмм

```
…
push 2
push 3
call proc1        ;ax=5
add sp,4
…
proc1 PROC
   mov bp,sp
   add bp,2
   mov ax,[bp]
   add bp,2
   add ax,[bp]
   ret
proc1 ENDP
```

Заносим в стек параметры

Вызываем proc1 команда call заносит в стек адрес следующей команды: “add sp,4”

Увеличиваем bp. Теперь он указывает на предпоследний элемент, т.е. «3»

Теперь в bp адрес последнего элемента стека, т.е. адреса возврата

Увеличиваем bp. Теперь он указывает на «2»

Возвращаемся по адресу, который по прежнему записан в стеке последним

Стек и организация подпрограмм

Соглашение вызова определяет следующие особенности процесса использования подпрограмм:

Соглашение вызова

- Расположение входных параметров подпрограммы и возвращаемых ею значений.
- Порядок передачи параметров. Кто возвращает указатель стека на исходную позицию.
- Какой командой вызывать подпрограмму и какой — возвращаться в основную программу. Например, в стандартном режиме x86 подпрограмму можно вызвать через call near, call far и pushf/call far (для возврата применяются соответственно retn, retf, iret).
- Содержимое каких регистров процессора подпрограмма обязана восстановить перед возвратом.

Стек и организация подпрограмм

Соглашение вызова cdecl

Соглашение вызова

Основной способ вызова для Си (отсюда название, сокращение от «c-declaration»). Аргументы передаются через стек, справа налево. Очистку стека производит **вызывающая** программа. Возвращаемое значение записывается в регистр ax (eax).

Пример:

```
int function_name(int, int, int);
int a, b, c, x;
x = function_name(a, b, c);
```

```
push c ; arg 3
push b ; arg 2
push a ; arg 1
call function_name
add esp, 12
mov x, eax
```

Очистка стека

Стек и организация подпрограмм

Соглашение вызова

Соглашение вызова pascal
Основной способ вызова для Паскаля. Аргументы передаются через стек, слева направо. Указатель стека на исходную позицию возвращает подпрограмма. У функций неявно создаётся дополнительный первый изменяемый параметр Result, через который и возвращается значение.

Соглашение вызова stdcall/winapi
Применяется при вызове функций WinAPI. Аргументы передаются через стек, справа налево. Очистку стека производит **вызываемая** подпрограмма.

Соглашение вызова fastcall
Передача параметров через регистры. Если все параметры и промежуточные результаты умещаются в регистрах, манипуляции со стеком вообще не нужны. **Fastcall** не стандартизирован, поэтому используется только в функциях, которые программа не экспортирует наружу.