Тема 9

Процедуры в языке Ассемблера

Понятие процедуры

В языке Ассемблера для обозначения логических модулей программы используется более общий термин *процедура*.

Нестрого *процедуру* можно определить как именованный блок команд, оканчивающийся оператором возврата.

Оформление процедур

Для оформления процедуры используется конструкция

```
имя_процедуры PROC    [NEAR|FAR]
                ;тело процедуры
                RET
имя_процедуры ENDP
```

Следует сказать, что, в отличие от записи процедур на языках высокого уровня, директивы **PROC** и **ENDP** не генерируют никакого кода.

Поэтому программист сам должен следить за тем, чтобы вход в процедуру не был выполнен после выполнения команды, непосредственно предшествующей процедуре, а выход из процедуры выполнялся только командой **RET.**

Работа с процедурами

Изучение механизма реализации процедур предполагает рассмотрение следующих вопросов:

- как осуществить вызов процедуры и выход из нее;
- как передать в процедуру параметры.

Команды для работы с процедурами

- **`CALL адрес`** – вызов процедуры

  `адрес` – точка входа в процедуру.

  Для указания типа вызова (ближний или дальний) можно задать спецификатор **`NEAR`** или **`FAR`**

- **`RET`** – выход из процедуры

Схема вызова процедур

Различают:

- ближний (**NEAR**) и дальний (**FAR**) вызовы (но не короткий!);
- прямой и косвенный вызовы.

Первый вопрос решен аппаратно, второй – программно.

Получают:

- прямой ближний;
- прямой дальний;
- косвенный ближний;
- косвенный дальний.

Схема вызова процедур

Тип адресации при вызове процедуры зависит от используемой модели памяти.

Директива **`.model`** автоматически устанавливает нужный атрибут для вызываемых процедур:

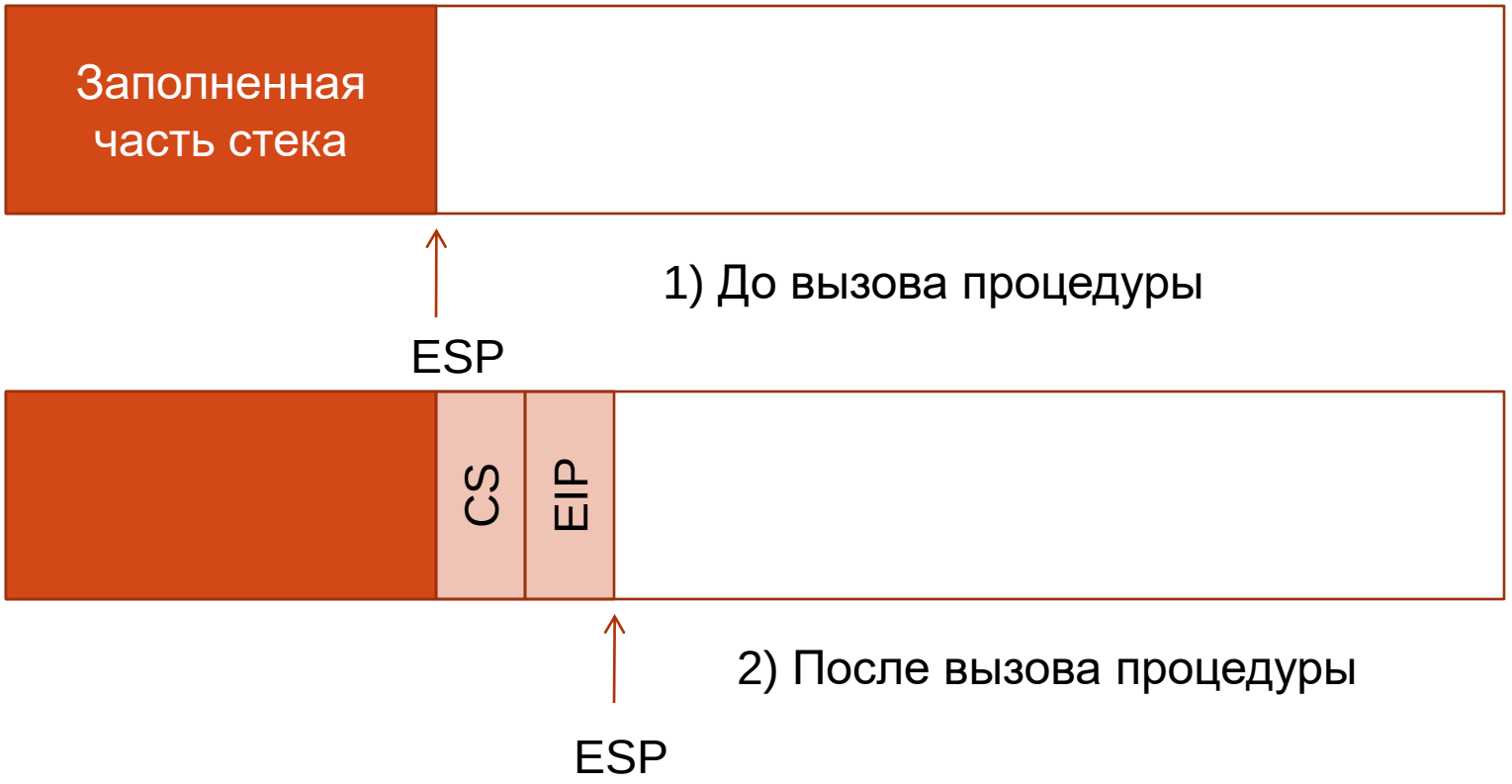
- модели **`tiny`**, **`small`** и **`compact`** устанавливают атрибут **`near`**,
- модели **`medium`**, **`large`** и **`huge`** — атрибут **`far`**.
- для 32-разрядных приложений, использующих модель **`flat`**, все вызовы процедур считаются ближними (**`near`**).

Схема вызова процедур (продолжение)

Вызов процедуры связан с изменением регистров **CS** и **EIP/IP** (или только второго регистра).

Перед изменением регистров **CS** и **EIP/IP** в стеке сохраняются их старые значения. При возврате из процедуры эти значения извлекаются из стека.

Работа со стеком при дальнем вызове процедур



После этого в регистры **`CS/EIP`** заносится информация либо из команды (*прямой вызов*), либо из области памяти, адрес которой определяется в команде (*косвенный вызов*).

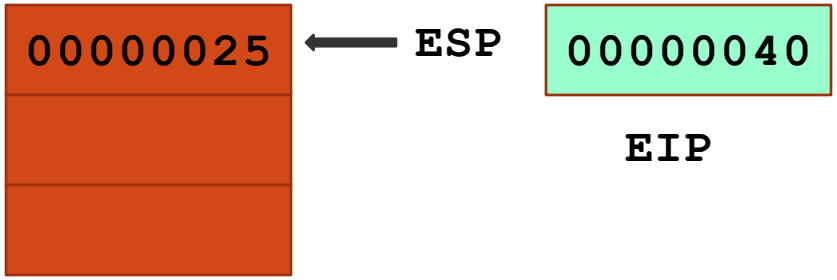
Правила для работы с процедурами

- каждой выполненной команде **CALL** должна соответствовать команда **RET** и наоборот;
- значения регистра **ESP** в начале работы процедуры и перед выходом из нее должны совпадать.

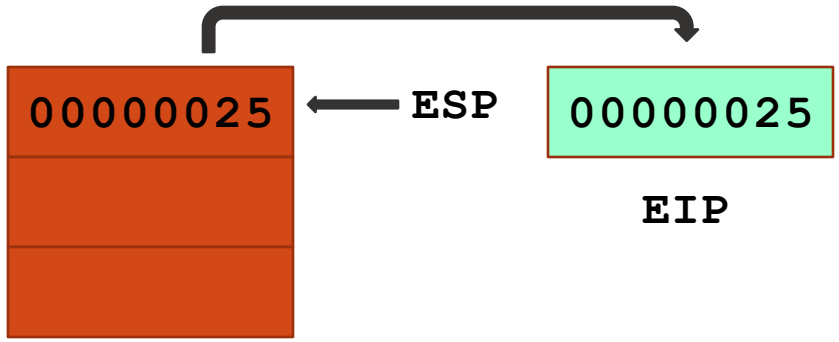
Пример вызова и возврата из процедуры

```
          main PROC
. . .
00000020 call MySub
00000025 mov  eax, ebx
. . .
    MySub PROC
00000040 mov  eax, edx
. . .
          ret
    MySub ENDP
```

Вызов



Возврат



Варианты размещения процедур в программе

- *в начале программы* (до первой исполняемой команды);
- *в конце программы* (после команды, возвращающей управление операционной системе);
- *промежуточный вариант* — тело процедуры располагается внутри другой процедуры или основной программы (в этом случае необходимо предусмотреть обход процедуры с помощью команды безусловного перехода **`JMP`**);
- *в другом модуле*.

Примеры записи процедур (реальный режим работы процессора)

Вариант 1 – процедура записана до головной программы

```
                .model small
                .stack 100h
                .data
crlf            db    13, 10, '$'
                .code
writeln         proc
                mov   dx, offset crlf
                mov   ah, 09h
                int   21h
                ret
writeln         endp
                .startup
                call  writeln
                .exit
end @startup  ; явно указываем точку входа в программу
```

Примеры записи процедур (реальный режим работы процессора)

Вариант 2 – процедура записана после головной программы

```
                .model small
                .stack 100h
                .data
crlf            db    13, 10, '$'
                .code
                .startup
                call  writeln
                .exit
writeln         proc
                mov   dx, offset crlf
                mov   ah, 09h
                int   21h
                ret
writeln         endp
end
```

Примеры записи процедур (реальный режим работы процессора)

Вариант 3 (неверный) – процедура будет выполнена дважды!

```
                .model small
                .stack 100h
                .data
crlf            db    13, 10, '$'
                .code
                .startup
                call  writeln
writeln         proc
                mov   dx, offset crlf
                mov   ah, 09h
                int   21h
                ret
writeln         endp
                .exit
end
```

Работа с локальными данными

```
writeln proc
        jmp    @@m1
@@crlf  db     13, 10, '$'
@@m1:
        push   ds
        push   cs
        pop    ds
        mov    dx, offset @@crlf
        mov    ah, 09h
        int    21h
        pop    ds
        ret
writeln endp
```

Передача параметров в процедуры

В архитектуре процессоров x86 нет никаких аппаратных решений по механизму передачи параметров. Это означает, что программист сам может реализовать любой механизм передачи параметров, приемлемый для него.

К настоящему времени сложились определенные стандарты, следование которым весьма желательно!

Параметры могут быть переданы:

- через общие области памяти;
- через регистры;
- через стек.

Передача параметров через общие области памяти

Аналог в языках высокого уровня – использование глобальных переменных.

Суть метода – и головная программа, и процедура работают с одной и той же областью памяти. Для доступа к этой общей области используются метки, которые в дальнейшем компилируются в конкретные адреса.

Достоинство – простота реализации.

Недостаток – если процедура должна обрабатывать различные данные при различных вызовах, на головную программу возлагается дополнительная обязанность по пересылке данных в общие области или из них.

Передача параметров через регистры

Суть метода – перед обращением к процедуре в регистры общего назначения заносится информация, необходимая для работы процедуры. В этом случае при описании программы необходимо подробно и четко указать, какие параметры через какие регистры передаются.

Достоинство – простота реализации и бОльшая гибкость по сравнению с первым способом.

Недостаток – количество регистров общего назначения невелико, и их может не хватить для передачи большого количества параметров.

Пример передачи параметров через регистры

Требуется написать процедуру сложения двух целых чисел. Использовать передачу параметров через регистры.

```
; слагаемые передаются через регистры eax и ebx
; сумма возвращается в eax
 SUM  proc
      add eax,ebx
      ret
 SUM  endp
;  вызов процедуры
      mov eax,A
      mov ebx,B
      call  SUM
      mov   S,eax
```

Передача параметров через стек

Суть метода – вызывающая программа перед выполнением команды **CALL** заносит в стек либо значения фактических параметров, либо их адреса.

Первый способ соответствует передаче параметров по значению, второй – по адресу.

Процедура извлекает из стека полученную информацию без использования команд **POP**, используя тот факт, что к содержимому стека можно обращаться, как к любому другому участку памяти.

Достоинство – количество параметров и их размер практически неограниченны.

Недостаток – необходимо продумывать механизмы помещения в стек и извлечения из стека.

Схема передачи параметров через стек

Рассмотрим вызов NEAR-процедуры с двумя параметрами:

```
push eax ;  второй параметр
push edx ;  первый параметр
call myproc
```

Содержимое стека после **`call myproc`**:

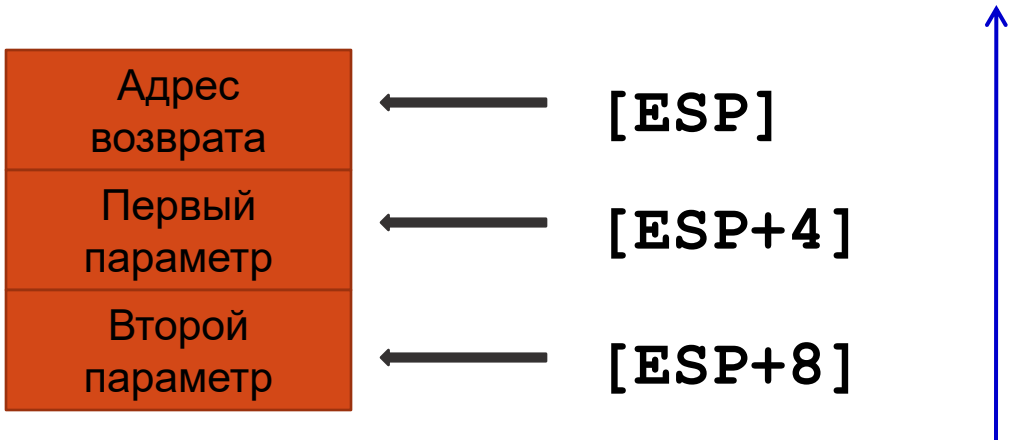
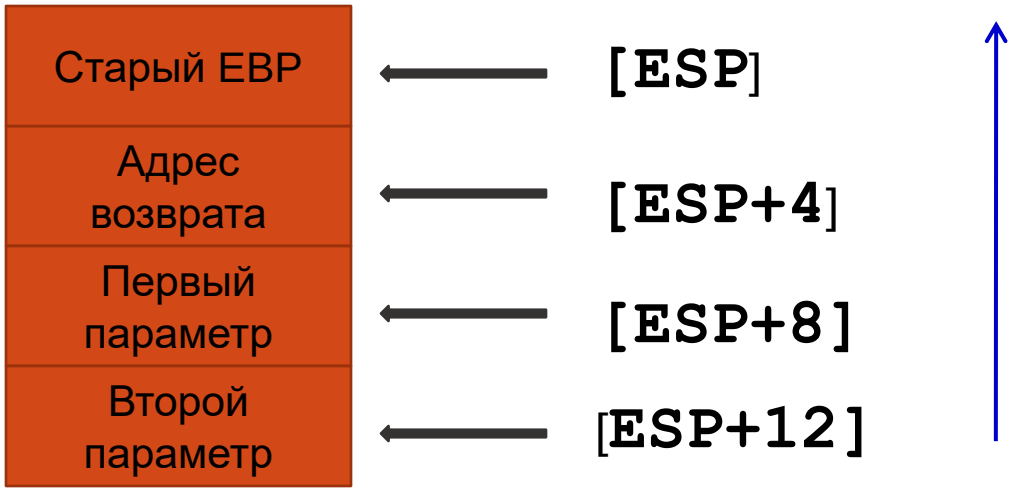


Схема передачи параметров через стек

Фрагмент процедуры:

```
myproc  proc
        push ebp
        mov  ebp, esp
        mov  eax, dword ptr [esp+8] ;первый параметр
        mov  ebx, dword ptr [esp+12];второй параметр
. . .   pop  ebp
myproc endp
```

Стек после **`push ebp`**:



Проблемы при передаче параметров через стек

При передаче данных через стек необходимо решить следующие вопросы:

- в каком порядке передавать параметры;
- какая программа должна очищать стек.

Возможны два варианта очистки стека:

- процедура очищает стек в момент выхода (выполнения команды RET);
- вызывающая программа очищает стек после того, как она вновь получила управление.

Очистка стека процедурой

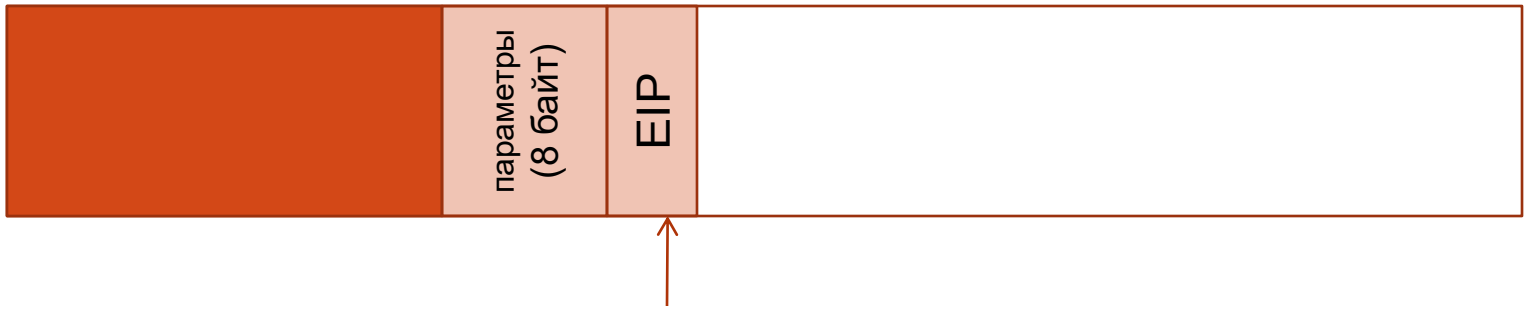
Этот способ реализован аппаратно: общий формат команды **RET** включает непосредственный операнд – число байт, которые выталкиваются из стека при выполнении команды **RET**:

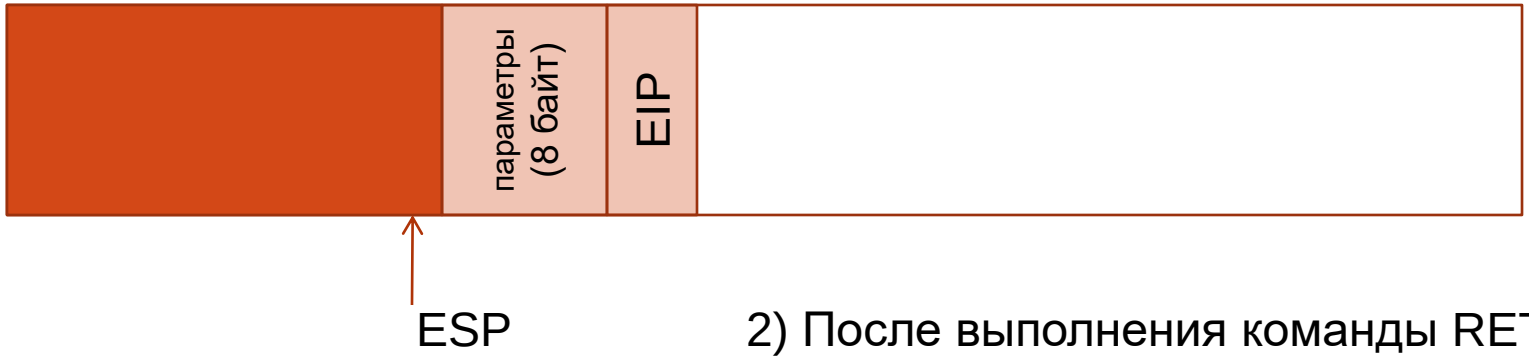
```
RET количество_байтов
```

Недостатки:

- размер блока параметров должен быть определен в момент написания программы, и это не позволяет, например, передавать переменное число параметров;
- в стеке нельзя передавать возвращаемые значения.

Иллюстрация передачи параметров через стек





Очистка стека головной программой

Этот способ должен быть реализован программно:

1) предполагается, что поле вызова процедуры головная программа должна чистить стек, например:

```
mov eax, x
mov edx, y
call myproc
pop edx
pop eax
```

2) можно откорректировать регистр указателя стека **ESP** на величину *4•n*, где *n* — количество аргументов, например:

```
call  my_prog
mov   eax, [esp] ;извлечение выходных параметров
add   esp, 8
```