12. Передача параметров через стек. Локальные данные подпрограмм

Этот способ важен для следующих случаев:

- рекурсивные вызовы в ассемблерных программах
- передача параметров при вызове ассемблерных программ из программ на языках высокого уровня
- вызов API-функций в ассемблерных Windows-программах

Необходимые основные шаги:

- основная программа перед вызовом подпрограммы помещает в стек входные значения
- вызванная подпрограмма извлекает значения из стека и использует их
- после завершения работы подпрограммы производится очистка стека для восстановления его состояния до вызова подпрограммы.

Взаимодействие подпрограмм может выполняться двумя способами – в так называемом Pascal-стиле и C-стиле.

Особенности Pascal-стиля:

- входные параметры записываются в стек в их естественном порядке, т.е. так, как они объявлены в заголовке - слева направо, после чего в вершине стека оказывается самый последний параметр
- за очистку стека отвечает подпрограмма

Особенности C-стиля:

- параметры в стек записывается в обратном порядке: сначала последний, потом предпоследний и т.д., в итоге на вершине оказывается первый параметр
- очистку стека выполняет основная программа после возврата из подпрограммы

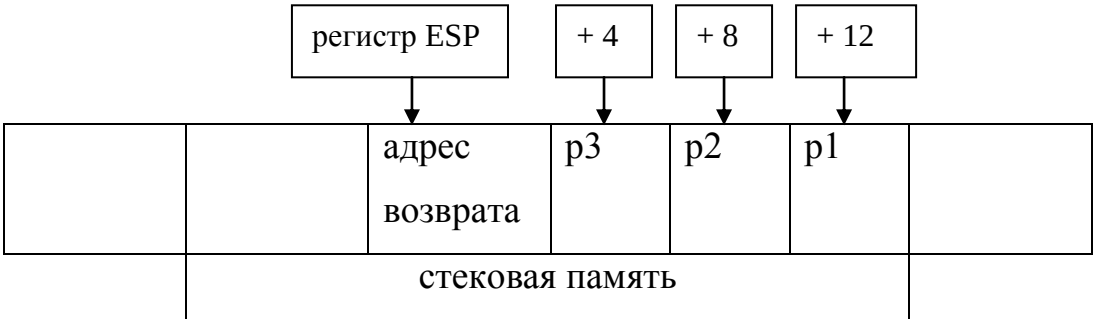
В качестве примера далее рассматривается способ взаимодействия в Pascal-стиле. Пусть в подпрограмму надо передать 3 параметра p1, p2 и p3. Тогда типичный фрагмент кода основной программы будет:

```
PUSH   p1
PUSH   p2
```

PUSH p3

CALL ИмяПП

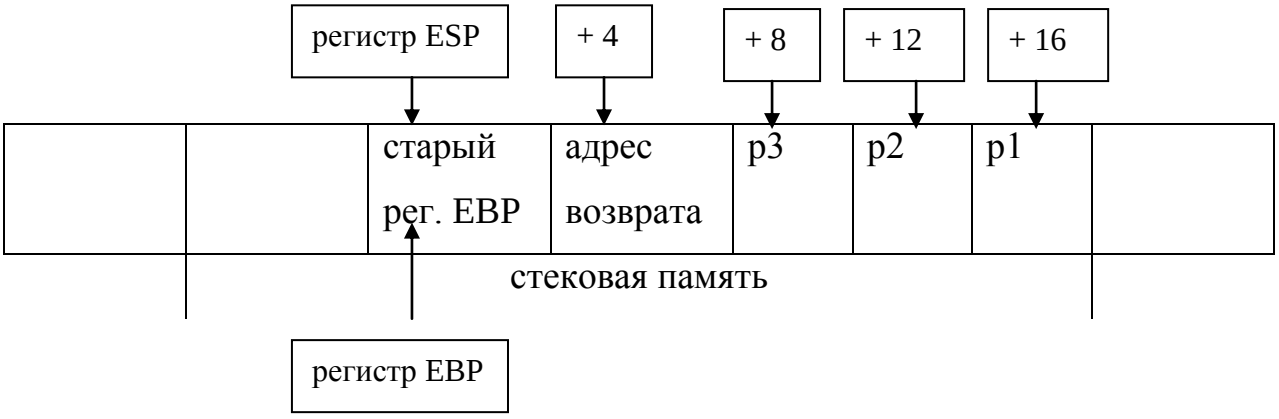
Отметим, что команда CALL запишет в вершину стека адрес возврата, и тем самым состояние стека при входе в подпрограмму будет следующим:



После вызова подпрограмме нужны параметры, но использовать команду POP нельзя, т.к. при выталкивании пропадает адрес возврата. Приходится использовать регистр EBP, установив в него значение из ESP и используя выражение вида [EBP + i]. Но регистр EBP может использоваться в основной программе, поэтому предварительно его надо сохранить в стеке, а потом уж использовать. Для всех этих операций в начале подпрограммы оформляется так называемый “пролог”:

```
MyProc    PROC
          PUSH   EBP          ; (EBP) → стек
          MOV    EBP,  ESP    ; (EBP) = вершина стека
```

После выполнения этих действий состояние стека будет следующим:



Тогда [EBP + 8] – адрес последнего параметра (третьего), [EBP + 12] – предпоследнее (второго), а [EBP + 16] – первого. Эти выражения можно

использовать в теле подпрограммы для выполнения необходимых действий с параметрами. Регистр EBP при этом изменяться не должен.

В конце подпрограммы надо выполнить “выходные” действия, т.е. оформить так называемый “эпилог”. Прежде всего надо учитывать, что к моменту завершения подпрограммы стек должен быть в том же состоянии, что и в начале работы подпрограммы. Поэтому если подпрограмма использовала стек для своих внутренних нужд, надо вернуть его состояние в начальное, например – командой MOV ESP, EBP.

После этого надо считать с вершины старое значение регистра EBP, потом – адрес возврата и очистить стек от входных параметров. Для подпрограмм, взаимодействующих в Pascal-стиле можно использовать специальную разновидность команды выхода из подпрограммы вида RET N, где N – число байтов, автоматически удаляемых из стека после извлечения из стека адреса возврата.

Надо учитывать, что N – это число байтов, а так как параметры в стеке – это двойные слова, то для удаления K параметров надо задать N = 4*K. Тогда эпилог в Pascal-стиле будет следующим:

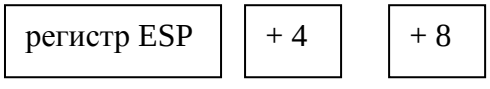
```
POP   EBP     ;  восстановим старое значение регистра EBP
RET   N       ;  удалить из стека K параметров и сделать возврат.
```

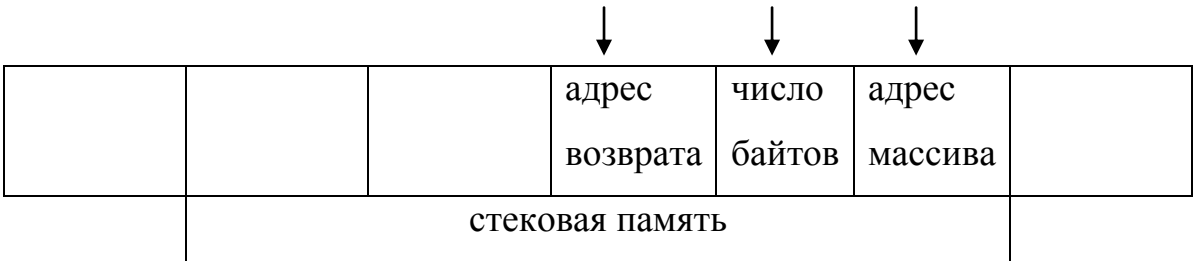
На эту простейшую схему часто накладывается необходимость восстановления сохраненных в стеке РОНов.

Пример. Оформить подпрограмму обнуления N байтов в массиве с начальным адресом Mas. Входные параметры - адрес массива и количество обнуляемых байтов – передаются через стек с помощью Pascal-модели. Фрагмент основной программы:

```
LEA    EAX,  Mas   ;  адрес массива – в регистр EAX
PUSH   EAX         ;  а потом из регистра – в стек
PUSH   100         ;  в стек – число обнуляемых байтов
CALL   NullMas     ;  вызов подпрограммы с именем NullMas
```

При входе в подпрограмму состояние стека будет следующим:

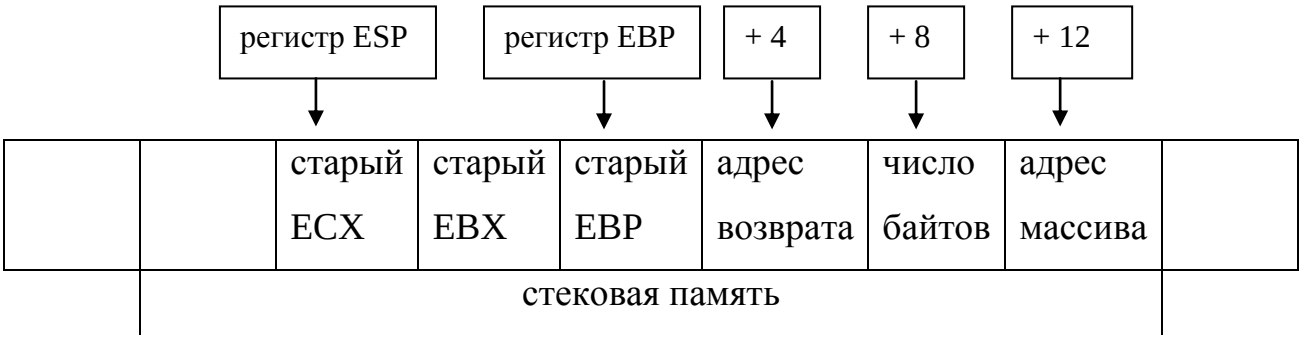




Для циклического обнуления байтов в массиве подпрограмма должна использовать два регистра – ECX как счетчик цикла и EBX в качестве хранилища адреса очередного байта массива (косвенная адресация). Поэтому при входе в подпрограмму значения этих регистров должны быть сохранены в стеке, а при завершении подпрограммы – восстановлены. Пролог в подпрограмме обнуления будет иметь следующий вид:

```
PUSH    EBP         ;  сохранить регистр EBP
MOV     EBP, ESP    ; загрузить в EBP адрес вершины стека
PUSH    EBX         ;  сохранить регистр  EBX
PUSH    ECX         ;  сохранить регистр  ECX
```

После этих операций состояние стека будет следующим:



Из этой схемы легко можно увидеть, как с помощью регистра EBP подпрограмма может получить доступ к обоим входным параметрам. Сама подпрограмма будет следующей:

```
NullMas  PROC
         "пролог"
         MOV     ECX, [EBP + 8]      ;  (ECX) = число обнуляемых байтов
         MOV     EBX, [EBP + 12]     ;  (EBX) = адрес массива
```

```
L:      MOV   BYTE  PTR  [EBX], 0  ; обнуление очередного байта
        INC   EBX                  ;  переход к следующему байту
        LOOP  L
        POP   ECX   ;  восстановление ECX
        POP   EBX   ;  восстановление EBX
        POP   EBP   ;  восстановление EBP
        RET   8     ;  возврат и удаление восьми байтов
NullMas ENDP
```

Довольно часто подпрограммам для выполнения своих функций требуется использовать временные переменные только на время своей работы. Как известно, такие переменные принято называть локальными. Если их немного, можно использовать регистры, предварительно сохранив их значения в стеке с последующим восстановлением. Если регистров недостаточно, можно использовать стек, выделяя в нем при запуске подпрограммы необходимую область и освобождая ее при завершении. В этом случае место под локальные переменные выделяется только на время работы подпрограммы.

При входе в подпрограмму со стеком надо сделать следующие операции:

- запомнить старое значение регистра EBP
- установить регистр EBP на вершину стека
- уменьшить значение в регистре ESP на количество байт, необходимое для хранения значений локальных данных

Например, если подпрограмме надо использовать две локальные переменные, то в начале подпрограммы необходимо:

```
PUSH   EBP
MOV    EBP, ESP
SUB    ESP, 8      ; уменьшить ESP на два двойных слова
```

Состояние стека после этого будет следующим:

Доступ к локальным переменным – по выражению [EBP – 4] или [EBP – 8]. В конце подпрограммы надо выполнить обратные действия:

```
MOV   ESP, EBP  ; отказ  от локальной памяти
POP    EBP      ; восстановить старое значение EBP
RET             ; или RET   N,  если необходимо
```

Данный механизм является основой реализации рекурсивных вызовов в ассемблерных программах. Главное правило - для локальных переменных рекурсивной подпрограммы использовать стек с правильной его очисткой при выходе. Кроме того, при каждом очередном входе в рекурсивную подпрограмму надо сохранять содержимое всех использующихся в подпрограмме регистров и восстанавливать их при выходе

Практические задания к теме №12.

Задание 1. Оформить подпрограмму обнуления N байтов в массиве с начальным адресом Mas. Входные параметры - адрес массива и количество обнуляемых байтов – передаются через стек