Учебный курс. Часть 25. Передача параметров через стек

Автор: xrnd | Рубрика: Учебный курс | 26-09-2010 |
Распечатать запись

В предыдущих частях учебного курса все параметры передавались процедурам через регистры. В этой статье мы рассмотрим другой способ — передачу параметров через стек. Часто этот способ оказывается удобнее. Через регистры можно передать максимум 6-7 параметров, а через стек — сколько угодно. Кроме того можно написать процедуру с переменным количеством параметров. (Подробнее о таких процедурах вы можете прочитать здесь.)

Если через регистры передаётся больше 2-3 параметров, то приходится сохранять регистры внутри процедуры и опять же использовать стек. С другой стороны, обращение к параметрам в стеке происходит медленнее. Если вы оптимизируете программу по скорости выполнения, то имеет смысл передавать параметры через регистры.

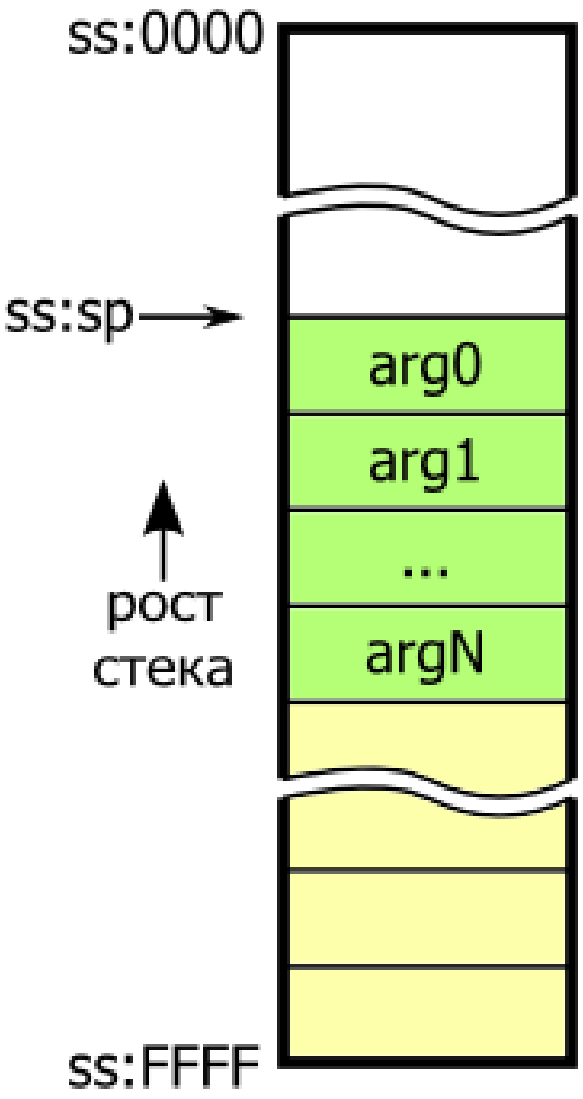
Помещение параметров в стек

Перед вызовом процедуры параметры необходимо поместить в стек с помощью команды PUSH. Здесь существует два варианта: параметры могут помещаться в стек в прямом или в обратном порядке. Обычно используется обратный порядок и его я буду использовать в примерах. Параметры помещаются в стек, начиная с последнего, так что перед вызовом процедуры на вершине стека оказывается первый параметр:

```
; Данные
arg0      dw 0
```

```
arg1      dw 12
argN      dw 345
;---------------------------------------------------------------
; Код
    push [argN]
    push ...
    push [arg1]
    push [arg0]
    call myproc
```

Перед выполнением команды CALL стек будет иметь следующий вид:

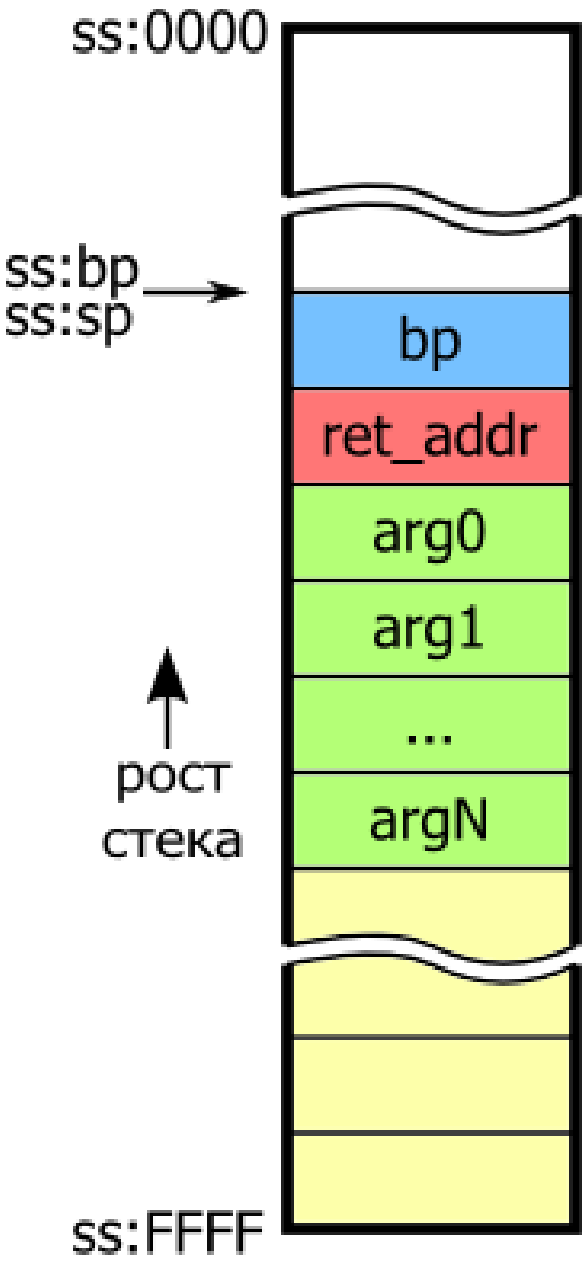


Обращение к параметрам внутри процедуры

Для обращения к параметрам внутри процедуры обычно используют регистр BP. В самом начале процедуры содержимое регистра BP сохраняется в стеке и в него копируется значение регистра SP. Это позволяет «запомнить» положение вершины стека и адресовать параметры относительно регистра BP.

```
;Процедура
myproc:
    push bp
    mov bp,sp
    ...
```

При выполнении кода процедуры стек будет иметь следующую структуру:



Здесь *ret_addr* обозначает адрес возврата, помещаемый в стек командой вызова процедуры, а *bp* — сохранённое значение регистра BP. В нашем случае стек имеет ширину 16 бит, поэтому первый параметр будет доступен как word[bp+4], второй как word[bp+6] и так далее.

```
    mov ax,[bp+4]       ;AX = arg0
    mov bx,[bp+6]       ;BX = arg1
    add ax,[bp+8]       ;AX = AX + arg2
```

Полезно представлять себе стек, чтобы правильно указывать смещения относительно регистра BP. Не забудьте перед возвратом из процедуры восстановить значение BP из стека.

Извлечение параметров из стека

После того, как процедура выполнилась, необходимо очистить стек, вытолкнув из него параметры. Тут тоже существует 2 способа: стек может быть очищен самой процедурой или кодом, который эту процедуру вызывал. Для первого способа используется команда RET с одним операндом, который должен быть равен количеству байтов, выталкиваемых из стека. В нашем случае он должен быть равен количеству параметров, умноженному на 2.

```
    push [arg1]
    push [arg0]
    call myproc
    ...
;-----------------------------------------------------------
;Процедура с двумя параметрами
myproc:
    push bp
    mov bp,sp
    ...
    pop bp
    ret 4         ;Из стека дополнительно извлекается 4 байта
```

Для второго способа нужно использовать команду RET без операндов. Стек восстанавливается после выполнения процедуры путём прибавления значения к SP. С помощью такого способа программируются процедуры с переменным количеством параметров. Процедура не знает, сколько ей будет передано параметров, поэтому очистка стека должна выполняться вызывающим кодом.

```
    push [arg1]
```

```
    push [arg0]
    call myproc2
    add sp,4                ;Восстановление указателя стека
    ...
;-----------------------------------------------------------------
;Процедура с двумя параметрами (не очищает стек)
myproc2:
    push bp
    mov bp,sp
    ...


    pop bp
    ret
```

Соглашения вызова

Совокупность таких особенностей, как способ и порядок передачи параметров, механизм очистки стека, сохранение определённых регистров в процедуре и некоторых других называется *соглашениями вызова*. Соблюдение этих соглашений является важным, если вы из своей программы обращаетесь к компонентам, написанным на других языках программирования, вызываете функции ОС, или хотите из других языков вызывать процедуры, написанные на ассемблере. В остальных случаях процедуры на ассемблере можете писать так, как вам больше нравится 🙂

Пример

В качестве примера рассмотрим процедуру с тремя параметрами: *a*, *b* и *c*. Процедура вычисляет значение выражения (*a*+*b*)/*c*. Параметры передаются через стек в обратном порядке, результат возвращается в регистре AX, стек восстанавливается вызываемой процедурой. Все числа — 16-битные целые со знаком.

```
1 use16                         ;Генерировать 16-битный код
```

```
 2 org 100h                    ;Программа начинается с адреса 100
 3     jmp start               ;Переход на метку start
 4 ;-------------------------------------------------------------
 5 ; Данные
 6 a     dw 81
 7 b     dw 273
 8 x     dw ?
 9 ;-------------------------------------------------------------
10 start:
11     push 3                  ;c=3
12     push [b]                ;b
13     push [a]                ;a
14     call primer             ;Вызов процедуры
15     mov [x],ax              ;x=(a+b)/c
16
17     mov ax,4C00h            ;\
18     int 21h                 ;/ Завершение программы
19
20 ;-------------------------------------------------------------
21 ;Процедура с тремя параметрами: a, b, c.
22 ;Вычисляет значение выражения (a+b)/c. Результат возвращается
23 primer:
24     push bp                 ;Сохранение регистра BP
25     mov bp,sp               ;BP=SP
26     push dx
27
28     mov ax,[bp+4]           ;AX=a
29     add ax,[bp+6]           ;AX=(a+b)
30     cwd                     ;DX:AX=(a+b)
31     idiv word[bp+8]         ;AX=(a+b)/c
32
33     pop dx
34     pop bp                  ;Восстановление регистра BP
35     ret 6                   ;Возврат с извлечением параметров
```

Как видите, всё довольно просто. Главное — не запутаться со смещениями относительно BP.

Упражнение

Напишите любую процедуру с 4-5 параметрами, передаваемыми через стек. Вызовите процедуру в своей программе. Запустите программу в Turbo Debugger и посмотрите, как происходит работа со стеком. Результаты можете писать в комментариях или на форуме.

Следующая часть »

Комментарии:

IgorKing
06-12-2010 18:05

Теперь тут. Как передавать адрес ,например строки, через стек? Точнее как обращатся к переменным по адресу, который лежит в стеке? А то вот так, как-то не так:

mov al,byte[(bp+8)+si]
cmp byte[(bp+4)+di],al

[Ответить]

xrnd
06-12-2010 22:32

Так, конечно, не получится 🙂 Адрес строки надо сначала поместить в регистр.

```
    mov bx,[bp+8]
    mov al,byte[bx+si]
```

[Ответить]

xrnd
06-12-2010 22:40

С двумя строками сложнее. Регистров-то мало.
Можно один адрес поместить в SI, а другой в DI и делать инкремент этих регистров. Либо ещё как-то выкручиваться:

```
    mov bx,[bp+8]
    mov dx,[bp+4]
    mov al,[bx+si]
    xchg bx,dx
    cmp [bx+di],al
```

[Ответить]

IgorKing
07-12-2010 13:19

Да, у меня там простейший поиск в строке. Пока (может поменяю алгоритм) нужно два адреса строк и два независимых слагаемых к адресам.

[Ответить]

Станислав
17-12-2010 16:13

Меня интересует как средствами bios или dos создать на жёстком диске файл и записать в него байты с определённым значением, так же процедура чтения файла, пожалуйста напишите подробный пример, нигде не могу найти, и изучение assembler у меня стоит на месте

[Ответить]

xrnd
17-12-2010 16:56

Средствами BIOS работать с файлами не получится. Можно только напрямую работать с жестким диском. Конечно, можно

расковырять структуру файловой системы и создать там файл… но это дело сложное.

Средствами DOS можно работать с файлами.
А не проще воспользоваться функциями Windows?

[Ответить]

Станислав
17-12-2010 17:06

Я получил Ваш ответ, конечно спасибо, но я знаю как работать с файлами в дельфи, меня интересует как это в ассемблере средствами dos, я конечно понимаю, что это не совеременно, но я уже год не могу сдвинуться в ассемблере с мертвой точки, потому как все справочники столь же информативны как и Ваш ответ, пожалуйста, наапишите мне конкретный пример как средствами dos создать на жёстком диске файл и записать в него байты с определённым значением, может быть Вы не понимаете, насколько это для меня имеет значение, у меня уже два десятка справочников, которые ходят вокруг ассемблера.

[Ответить]

xrnd
25-12-2010 00:31

Написал пример работы с файлами в DOS. Если что-то непонятно, пишите вопросы в комментариях к статье.

[Ответить]

Станислав
25-12-2010 07:29

Спасибо за помощь, я вот нашёл на просторах следующий код

```
.model tiny
.code
org 100h
start:
mov cx, 0
mov ax, 3c00h; вот у меня почему-то не работает прерывание 3Ch, может компилирую не правильно?
lea dx, file
int 21h
mov file_handle, ax
mov ah, 40h
mov bx, file_handle
mov si,6
mov cx, si
lea dx, x
int 21h
ret
file db ‘test.txt’,0
file_handle dw ?
x db ‘текст’,0
end start
```

А вот мне интересно, не вывести строку, а прочитать с определённого байта и записать в определённый файл, вот как мне быть?

[Ответить]

xrnd
25-12-2010 14:09

Исходник не очень хороший, но вроде должен работать.
Тут нет поверки ошибок, нет закрытия файла в конце. К тому же нулевой байт в конце строки x не нужен, если записывается текстовый файл.
Как не работает? Файл вообще не создаётся?

Для того, чтобы прочитать с определённого байта, нужно установить файловый указатель перед чтением. Это делается функцией 42h.

[Ответить]

Станислав
25-12-2010 19:33

Вот тот предыдущий исходник у меня работает а вот:

```
msg macro chars
mov ax, 0900h
lea dx, chars
int 21h
endm
endc macro
mov ax, 0100h
int 21h
mov ax, 4c00h
int 21h
endm
openFile macro nameFile, leaFile
mov ax, 3d00h
lea dx, nameFile
int 21h
jc open_error
mov leaFile, ax
jmp open_exit
open_error:
mov leaFile, 0
open_exit:
endm
createFile macro nameFile, leaFile
mov ax, 3c00h
lea dx, nameFile
int 21h
```

```
jc create_error
mov leaFile, ax
jmp create_exit
create_error:
mov leaFile, 0
create_exit:
endm
seekFile macro leaFile, iSeek
mov ax, 4200h
mov bx, leaFile
mov cx, 0
mov dx, iSeek
int 21h
endm
readFile macro leaFile, buff, eoff
mov ax, 3f00h
mov bx, leaFile
mov cx, eoff
lea dx, buff
int 21h
endm
writeFile macro leaFile, buff, eoff
mov ax, 4000h
mov bx, leaFile
mov cx, eoff
lea dx, buff
int 21h
endm
closeFile macro leaFile
mov ax, 3e00h
mov bx, leaFile
int 21h
endm
stk segment stack
db 256 dup(?)
stk ends
```

```
data segment
sErrorOpenFile db ‘Mistake at opening a file «1.txt»‘, ‘$’
sErrorCreateFile db ‘Mistake at creation of a file «2.txt»‘, ‘$’
File1 db ‘C:\1.txt’, 0h
File2 db ‘C:\2.txt’, 0h
leaFile dw ?
buf db 256 dup(‘$’)
data ends
code segment
assume cs: code, ds: data, ss: stk
main proc
mov ax, data
mov ds, ax
begin:
; открываем файл
openFile File1 leaFile
cmp leaFile, 0
je error_open
; указатель на 3-й байт
seekFile leaFile 2
; считываем 5-ть байтов
readFile leaFile buf 5
; закрываем файл
closeFile leaFile
; создаём файл 2
createFile File2 leaFile
cmp leaFile, 0
je error_create
; пишем в файл 5-ть байтов
writeFile leaFile buf 5
; закрываем файл
closeFile leaFile
endc
error_open:
msg sErrorOpenFile
endc
```

```
error_create:
msg sErrorCreateFile
endc
main endp
code ends
end main
```

Этот не работает, мне нужно прочитать файл весь, а потом с определённого байта записать весь в другой файл у меня TASM 5.0. Лучше всего, пожалуйста, на примере, а то как-то вокруг да около — не совсем то.

[Ответить]

xrnd
25-12-2010 20:34

У тебя уже все примеры есть, чтобы разобраться в этих файлах DOS.

Напиши хотя бы черновой вариант своей программы. Можешь для этого создать тему на форуме со своим кодом, а я помогу исправить ошибки.

Зачем тебе TASM? Он старый, давно не развивается и даже не поддерживается. Лучше на FASM учиться.

[Ответить]

Станислав
25-12-2010 20:53

Я как бы не помню, каким образом создавать новые разделы, а пользуюсь я TASM, как бы оно сложнее будет на FASM перейти, я этот не знаю, а по тому ещё и не найдёшь ничего. В основном все книги касаются masm b TASM, второй мне подходит, потому как готовая прога получается меньше

размером. Это основная причина, по которой мне нужен ассемблер. А вот раздел по работе с файлами уже создан, может поместишь туда пример как на TASM сделать программу, чтобы считать один файл, а потом перезаписать с определённого байта в другой. У меня тут одна практическая мысль есть с этим связанная.

[Ответить]

xrnd
25-12-2010 22:13

Я на TASM уже лет 5 не программировал. Да и примеров работы с файлами в DOS уже достаточно. Пытайся сам, я за тебя писать программу не буду 🙂

[Ответить]

argir
05-01-2011 17:09

Программа вычисления средней температуры:

```
use16
org 100h

push -25
push -12
push -2
push -8
push -3;допустим зима
push 5; количество дней
call sr_t
add sp,12; восстановление стека
mov ax,4C00h ;\
int 21h ;/
```

```
;————————————————————————————
;функция вычисляет среднее значение
;первый параметр в стеке-количество данных
;результат в ax
sr_t:
push bp
mov bp,sp
push dx
push cx
push si
xor ax,ax
mov cx,[bp+4]
jcxz pr2;если количество данных=0, то ошибка
mov si,6;смещение данных относительно вр в si
mov ax,[bp+si]
dec cx
sum: add si,2
add ax,[bp+si]
jo pr2;если переполнение, то ошибка
lp1: loop sum
cwd
idiv word[bp+4];вычисление среднего значения
jmp pr1
;ошибка ax=0,cf=1
pr2:
xor ax,ax
stc
pr1:
pop si
pop cx
pop dx
pop bp;восстановление регистров
ret
```

Я не смог разобраться как увеличить диапазон суммирования для чисел со знаком,

ведь все равно перед делением добавляем регистр dx. Вы обошли эту тему, а в интернете как-то все размыто.

[Ответить]

xrnd
11-01-2011 00:00

Хорошая программа. Даже с проверкой переполнения! 🙂

Увеличить диапазон суммирования можно, используя команды сложения или вычитания с переносом (см. часть 10).

Например, так:

```
    mov ax,[bp+si]
    cwd    ;DX:AX = AX
    dec cx
sum: add si,2
    add ax,[bp+si]
    adc dx,0
    loop sum
    idiv word[bp+4] ;вычисление среднего значения
```

В этом случае переполнение можно не проверять, так как его здесь не будет.

[Ответить]

argir
12-01-2011 10:30

А как при сложении отрицательных чисел учитывать если происходит перенос в знаковый разряд? Или это делает команда adc dx,0 автоматически?

[Ответить]

xrnd
13-01-2011 16:49

Извиняюсь, такой код будет правильно работать только для чисел без знака.
Можно считать сумму в двух других регистрах или в локальной переменной, а DX:AX использовать для преобразования слова в двойное слово.

```
    mov ax,[bp+si]
    cwd
    add bx,ax
    adc di,dx    ;Сумма в DI:BX
```

[Ответить]

Гость
05-02-2011 19:50

use16
org 100h
push ’01’
push ’02’
push ’03’
push ’04’
push ’05’
push ‘$$’
mov bp,sp ; фиксируем конец данных
;—-Передача параметров через стёк
push rez ; адрес строки для отображения
push 9 ;
push 1 ;
push 6 ;
;————————
pop cx
pop di

```
pop si
zl:
mov al,byte[bp+di] ; перемешаемся по значениям хоронящемся в тёске
mov [rez+si],al ; перемещаемся по памяти от конца данных
add di,2
dec si
loop zl
pop [dxa]
call print_str
mov ax,4C00h
int 21h
;-------------------
rez rb 10
dxa rw 1
print_str:
mov ah,09h
mov dx,[dxa]
int 21h
mov ah,08h
int 21h
ret
```

[Ответить]

xrnd
09-02-2011 17:26

А где же собственно процедура? 🙂
Если вызвать её командой CALL, то нельзя будет извлечь параметры с помощью POP, так как в стеке будет лежать ещё адрес возврата.
Кстати, в массиве rez перед началом строки будет 4 нулевых байта.

[Ответить]

Гость
10-02-2011 12:35

Спасибо за разнесение , а так можно ?

```
use16
org 100h
push ’01’
push ’02’
push ’03’
push ’04’
push ’05’
push ‘$$’
mov bp,sp
push rez
push 9
push 1
push 6
call pro
call print_str
mov ax,4C00h
int 21h
popz rw 1
;————————————-
rez rb 10
dxa rw 1
poz rw 1
print_str:
mov ah,09h
mov dx,[dxa]
int 21h
mov ah,08h
int 21h
ret
pro:
pop [poz] ; выгружаем адрес возврата
pop cx
```

```
pop di
pop si
zl:
mov al,byte[bp+di]
mov [rez+si],al
add di,2
dec si
loop zl
pop [dxa]
add[dxa],4 ; убераем первые 4 пустых байта
push [poz]
ret
```

[Ответить]

xrnd
10-02-2011 18:35

Можно как угодно 🙂 Это же ассемблер.

Но обычно так не делают. Твоей процедуре проще было бы передать параметры через регистры. С адресом возврата ты разобрался, временно поместив его в переменную. Но как в такой процедуре сохранить используемые регистры?

К тому же ты используешь глобальные переменные для сохранения значений внутри процедуры. Это не совсем хорошо. Во-первых, код становится запутанным. Во-вторых, такая процедура не может быть рекурсивной или вызываться из разных потоков.

[Ответить]

Гость
18-02-2011 20:19

У меня вопрос о регистре bp .
Можно его использовать по с ваему усмотрению в программе или это черевато.
Например регистр sp нельзя использовать как захочется , это понятно.
А вот bp как то не совсем понятно.

[Ответить]

xrnd
22-02-2011 20:07

Использовать BP по своему усмотрению можно.
Однако, тогда будет трудно работать с параметрами в стеке и локальными переменными.
Также надо учитывать, что BP по умолчанию адресует память в сегменте стека (относительно SS).

[Ответить]

RoverWWWorm
07-03-2011 14:21

```
;процедура PowerA234(A,b,c,d), вычисляющая вторую, третью и
;четвертую степень числа A и возвращающую эти степени
;соответственно в переменных b,c,d

;вопрос как возратить степени в переменные b, c, d ?

use16
org 100h

jmp start

a dw 4
b dw ?
```

```
c dw ?
d dw ?

start:
push [d]
push [c]
push [b]
push [a]

call PowerA234

;?

mov ax,4C00h
int 21h

PowerA234:
push bp
mov bp,sp
push cx
push si

mov cx,3
mov ax,word[bp+4] ; ax=a
mov si,6
label1:
imul word[bp+4] ;ax=ax*a

mov word[bp+si],ax ; b=a*a, c=a*a*a, d=a*a*a*a
add si,2

loop label1

pop si
pop cx
pop bp
ret 8
```

[Ответить]

xrnd
08-03-2011 14:22

Чтобы так сделать, нужно помещать в стек не значение переменной, а её адрес.

```
    push d
    push c
    push b
```

Далее в процедуре можно загрузить адрес в регистр и записать значение переменной по этому адресу.
Примерно так:

```
    mov bx,[bp+x]    ;x - смещение параметра в стеке
    mov [bx],ax      ;в ax значение, записываемое в переменную
```

В других языках программирования это называется передачей параметров по ссылке (через указатель), в отличие от передачи по значению, когда значение переменной копируется в стек.

Кстати, если не ошибаюсь, параметры должны передаваться этой процедуре в обратном порядке.

[Ответить]

RoverWWWorm
08-03-2011 16:08

;процедура PowerA234(A,b,c,d), вычисляющая вторую, третью и
;четвертую степень числа A и возвращающую эти степени
;соответственно в переменных b,c,d

```
use16
org 100h

jmp start

a dw 4
b dw ?
c dw ?
d dw ?

start:
push d
push c
push b
push [a]

call PowerA234
mov ax,[b] ;значения поместил в регистры чтоб, проверит
mov bx,[c] ;результат в отладчике
mov cx,[d]

mov ax,4C00h
int 21h

PowerA234:
push bp
mov bp,sp
push cx
push si
push bx

mov cx,3
mov ax,word[bp+4] ; ax=a
mov si,6
label1:
imul word[bp+4] ;ax=ax*a
mov bx,[bp+si]
```

```
mov [bx],ax ; b=a*a, c=a*a*a, d=a*a*a*a
add si,2

loop label1

pop bx
pop si
pop cx
pop bp
ret 8
```

[Ответить]

xrnd
09-03-2011 23:01

Всё правильно, замечательная программа.

[Ответить]

plan4ik
06-04-2011 20:54

hotel golovolomku no v spravo4nike usnal 4to RET prinimat toka ‘integer’ zna4enie ;((

```
use16
org 100h
jmp Start

arr dw 0x11, 0x22, 0x33, 0x44
dw 0x55, 0x66, 0x77, 0x88, 0

Start:
xor si, si
lp:
mov ax, [arr+si]
```

```
cmp ax, 0
je get_len
add si, 2
push [arr+si]
jmp lp

get_len:
call get_len_of_arr

mov ax, 4c00h
int 21h

; procedura pods4eta elementov array'a
; bp+4 == array pointer
; cx == count of elements
; 0 == end of array
get_len_of_arr:
push bp
mov bp, sp
xor cx, cx
mov si, 4
calc:
mov ax, [bp+si]
cmp ax, 0
je calc_exit
add si, 2
inc cx
jmp calc

calc_exit:
sub si, 4
pop bp
ret 0x10
;ret si ; obidno 4to on tak ne ho4et
```

[Ответить]

plan4ik
06-04-2011 21:01

popravka: v tzikle ‘calc’ add si. 2 nujno bilo delat pered proverkoy na 0, a izna4alno si stanovit v 2 …

[Ответить]

xrnd
08-04-2011 20:33

Интересная идея — считать длину массива, помещая все его элементы в стек 😀
В таком случае лучше передать в процедуру адрес массива.

Команды ret si конечно не может быть 🙂
Но без неё легко обойтись — нужно просто очищать параметры из стека в вызывающем коде. А в процедуре использовать ret без операндов.

[Ответить]

plan4ik
14-04-2011 23:43

так это … задание было таким 😉 «Напишите любую процедуру с 4-5 параметрами, передаваемыми через стек» — вот я и передал через стёк 😉 главное возможность а все остальное маневры))

[Ответить]

xrnd
15-04-2011 00:27

Так можно было и код процедуры передать через стек 😉

[Ответить]

алекс
28-06-2012 08:59

```
;программа вычисления определителя матрицы 2x2
;числа 16 бит, целые, со знаком
;-----------------------------------------
use16
org 100h
jmp start
X1.1 dw 12
X2.2 dw 15
X2.1 dw -48
X1.2 dw 86
buf rb 7
press db 13,10,'Press any key…$'
erms db 'overflow…',13,10,'$'

start:
push [X1.1]
push [X2.2]
push [X1.2]
push [X2.1]
call matr
call dig_out
jmp fin
error:
mov ah,09h
mov dx,erms
int 21h
fin:
mov ah,09h
mov dx,press
int 21h
mov ah,08h
```

```
int 21h
mov ax,4c00h
int 21h
;-------------------------------------------
matr:
push bp
mov bp,sp
push bx
mov ax,[bp+4]
imul word[bp+6]
jo error
mov bx,ax
mov ax,[bp+8]
imul word[bp+10]
jo error
sub ax,bx
pop bx
pop bp
ret 8
;-------------------------------------------
dig_out:
push bx
push dx
push si
push cx
mov bx,10
xor si,si
xor cx,cx
test ax,8000h
jz loop1
mov [buf+si],'-'
neg ax
inc si
loop1:
xor dx,dx
div bx
```

```
inc cx
add dx,’0′
push dx
test ax,ax
jnz loop1
loop2:
pop ax
mov [buf+si],al
inc si
loop loop2
mov [buf+si],’$’
mov ah,09h
mov dx,buf
int 21h
pop cx
pop si
pop dx
pop bx
ret
;————————————————————
```

[Ответить]

юля
13-04-2015 15:36

В примере вы в стек помещали регистр dx, а затем выталкивали из стека. Зачем нам вообще он нужен был? Спасибо за ответ

[Ответить]

Madlax
14-12-2015 02:28

а как передать и вернуть параметры массива который вводится с клавиатуры, после чего в функции он на пример сортируется как вернуть результат сортировки, а то не могу разобраться

[Ответить]

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

Уведомить меня о новых комментариях по email.

Уведомлять меня о новых записях почтой.