Учебный курс. Часть 26. Локальные переменные

Автор: xrnd | Рубрика: Учебный курс | 17-10-2010 |
Распечатать запись

До этой части учебного курса все переменные в наших программах были только *глобальными* — они создавались и инициализировались при запуске программы и к ним можно было обратиться из любой её части.

Локальные переменные используются для хранения промежуточных результатов во время выполнения процедуры. В отличие от глобальных, эти переменные являются временными и создаются при запуске процедуры. Для локальных переменных существует понятие *области видимости* — так называется область программы, в которой доступна переменная. Обычно в ассемблере область видимости ограничена процедурой, создавшей локальную переменную. Хотя возможны и более сложные варианты 😉

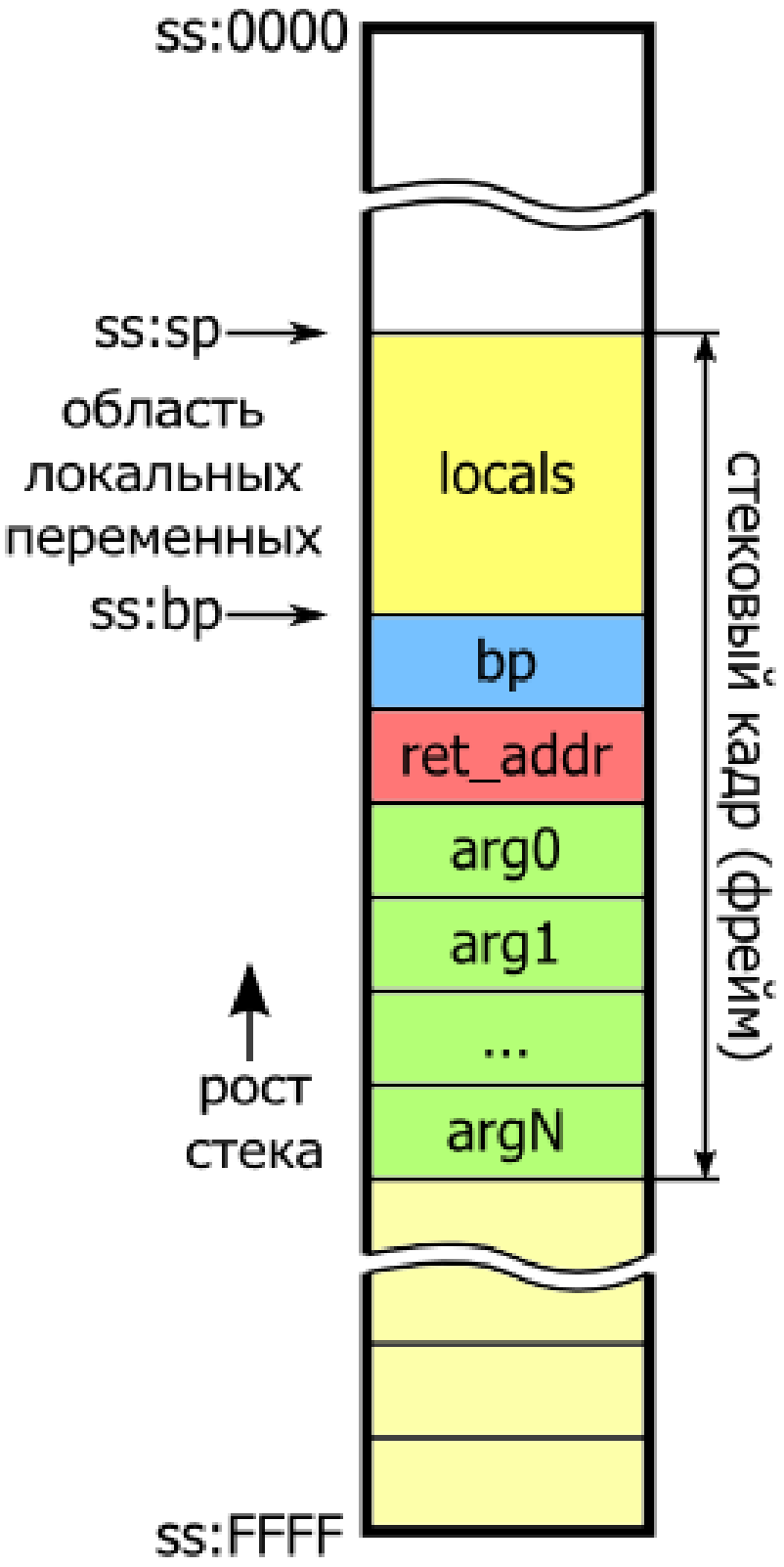
Создание локальных переменных

Чтобы создать локальные переменные в процедуре, необходимо выделить для них память. Эта память выделяется в стеке. Сделать это очень просто — достаточно вычесть из регистра SP значение, равное суммарному размеру всех локальных переменных в процедуре. Так как ширина стека равна 16 бит, то это значение должно быть кратно 2 байтам. При выходе из процедуры нужно восстановить указатель стека. Обычно это выполняется командой mov sp,bp (В bp сохраняется значение sp при входе в процедуру, как в случае с параметрами, передаваемыми через стек). Код процедуры с

локальными переменными будет выглядеть следующим образом:

```
;Процедура с локальными переменными
myproc:
    push bp                 ;Сохранение BP
    mov bp,sp               ;Копирование указателя стека в BP
    sub sp,locals_size      ;Выделение памяти для локальных перем
    ...
    mov sp,bp               ;Восстановление указателя стека
    pop bp                  ;Восстановление BP
    ret                     ;Возврат из процедуры
```

Код, выполняемый при входе в процедуру, называют также кодом *пролога*, а код, выполняемый при выходе, — кодом *эпилога*. После выполнения кода пролога стек будет иметь такой вид:



Область стека, включающая в себя параметры процедуры, адрес возврата, локальные переменные и сохранённые регистры, называется *кадром* или *фреймом* стека. Из рисунка понятно, что для обращения к локальным переменным внутри процедуры нужно использовать отрицательные смещения

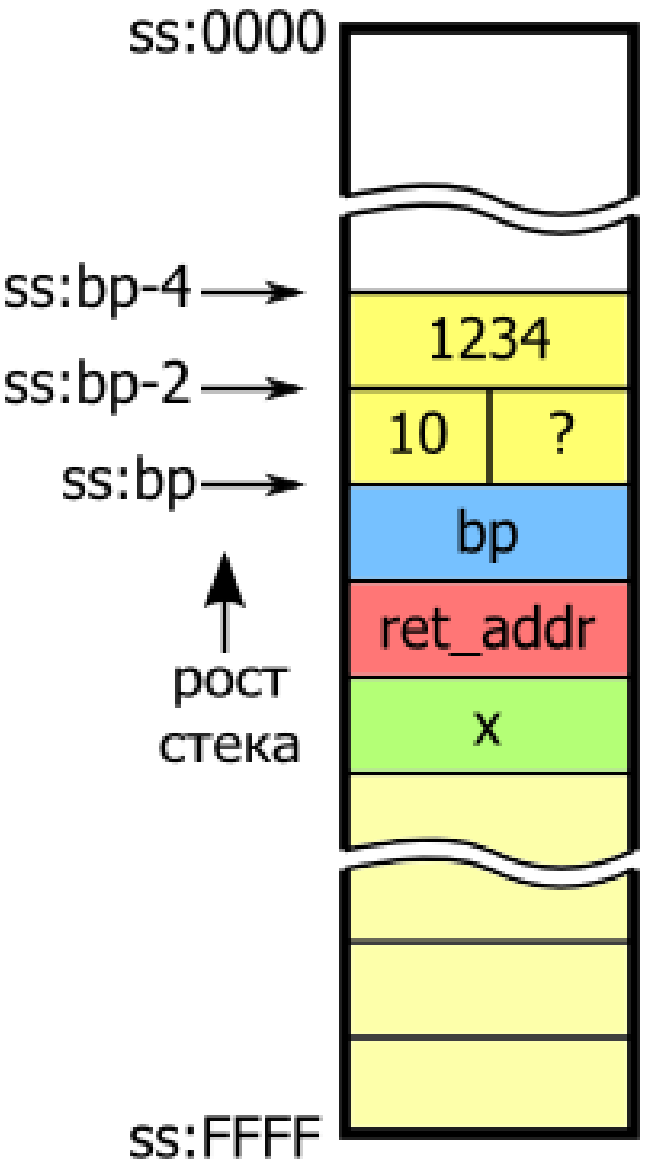
относительно регистра BP. Лучше всего показать это на примере 🙂

Пример процедуры с локальными переменными

Пример очень простой. Внутри процедуры создаются 2 локальных переменных: одна размером слово и вторая размером байт. Чтобы не нарушить стек, нужно выделить 4 байта (один байт не будет использоваться). Процедуре будет передаваться через стек один параметр — *x* (слово без знака). Она будет вычислять выражение 1234+*x*/10 и возвращать его в регистре AX. Числа 1234 и 10 будут нашими локальными переменными. Конечно, в данном случае можно обойтись и без локальных переменных, они здесь только для примера.

```
;Пример процедуры с двумя локальными переменными
;вход: параметр x в стеке
;выход: AX = вычисленное значение
simpleproc:
    push bp                 ;Сохранение BP
    mov bp,sp               ;Копирование указателя стека в BP
    sub sp,4                ;Выделение 4 байт для локальных перем

    mov word[bp-4],1234     ;Инициализация первой локальной перем
    mov byte[bp-2],10       ;Инициализация второй локальной перем

    mov ax,[bp+4]           ;AX = x
    div byte[bp-2]          ;AL = x/10
    xor ah,ah               ;AX = x/10
    add ax,[bp-4]           ;AX = 1234+x/10

    mov sp,bp               ;Восстановление указателя стека


    pop bp                  ;Восстановление BP
    ret 2                   ;Возврат из процедуры
```

После выполнения кода пролога первая локальная переменная будет находиться по адресу bp-4, а вторая по адресу bp-2. Обратите внимание, что стековые переменные должны быть явно инициализированы. Так как память выделяется в стеке, то изначально в них будет всякий мусор (а вовсе не нули). Структура стека при выполнении процедуры показана на рисунке:



Команды ENTER и LEAVE

В системе команд процессоров x86 существуют также специальные команды для работы с кадром стека процедуры: ENTER и LEAVE. Команда ENTER обычно размещается в

начале процедуры. У неё два непосредственных операнда: первый операнд — размер памяти, выделяемой под локальные переменные, второй операнд — уровень вложенности. в нашем случае второй операнд будет равен 0. Тогда по действию команда будет аналогична трём следующим командам:

```
    push bp                 ;\
    mov bp,sp               ; > или enter locals_size,0
    sub sp,locals_size      ;/
```

Второй операнд этой команды позволяет организовывать вложенные области видимости, как в некоторых языках высокого уровня. В ассемблере эти возможности используются редко.

Команда LEAVE не имеет операндов и аналогична по действию двум командам:

```
    mov sp,bp               ;\
    pop bp                  ;/ или leave
```

А так будет выглядеть наша процедура, если использовать команды ENTER и LEAVE:

```
;Пример процедуры с двумя локальными переменными
;Вход: параметр x в стеке
;Выход: AX = вычисленное значение
simpleproc2:
    enter 4,0               ;Создание кадра стека

    mov word[bp-4],1234     ;Инициализация первой локальной перем
    mov byte[bp-2],10       ;Инициализация второй локальной перем

    mov ax,[bp+4]           ;AX = x
    div byte[bp-2]          ;AL = x/10
    xor ah,ah               ;AX = x/10
    add ax,[bp-4]           ;AX = 1234+x/10
```

```
    leave                   ;Освобождение памяти, восстановление
    ret 2                   ;Возврат из процедуры
```

Вроде кажется, что проще, но компиляторы такой вариант практически не используют. Почему? Дело в том, что команда ENTER на современных процессорах выполняется гораздо медленнее, чем пролог из 3-х команд. Самый быстрый вариант такой:

```
;Процедура с локальными переменными
myproc:
    push bp                 ;Сохранение BP
    mov bp,sp               ;Копирование указателя стека в BP
    sub sp,locals_size      ;Выделение памяти для локальных перем
    ...
    leave                   ;Освобождение памяти, восстановление
    ret                     ;Возврат из процедуры
```

Упражнение

Напишите улучшенную процедуру для ввода десятичного числа (от 0 до 255) с консоли. Буфер для ввода строки должен быть локальным, то есть выделяться в стеке при вызове процедуры. Процедура должна вызываться без параметров и возвращать введённое число в регистре AL. Ваши результаты, а также вопросы можете писать в комментариях или на форуме.

Следующая часть »

Комментарии:

fufel
11-12-2010 18:31

Здравствуйте!

```
use16
org 100h
jmp start
;_____________________
vvod db «vvedite chislo ot 0 do 256:$»
ok db «OK!$»
oshbk db «Error!$»
pak db «Press any key…$»
end_str db 13,10,'$'
vved_chslo rb 1
;_____________________
start:
mov di,vvod
call prnt_str
call input_str
mov di,ok
call prnt_str
call end_line
mov di,pak
call prnt_str
mov ah,08h
int 21h
mov ax,4c00h
int 21h

input_str:
push bp
mov bp,sp
sub sp,6
mov byte[bp-6],4
mov dx,vvod
mov ah,09h
```

```
int 21h
mov dx,sp
mov ah,0ah
int 21h
call end_line
mov cl,byte[bp-5]
mov si,2
mov bh,10
xor ax,ax
push bp
sub bp,6
lp:
mov bl,byte[bp+si]
cmp bl,’0′
jl error
cmp bl,’9′
jg error
sub bl,’0′
mul bh
jc error
add al,bl
inc si
loop lp
pop bp

mov [vved_chslo],al
leave
ret

error:

mov di,oshbk
call prnt_str
call end_line
jmp start
```

```
prnt_str:
push ax
push dx
mov ah,09h
xchg dx,di
int 21h
pop dx
pop ax
ret

end_line:
push di
mov di,end_str
call prnt_str
pop di
ret
```

[Ответить]

xrnd
12-12-2010 21:50

Отличная программа! Локальный буфер для строки сделан правильно.
Проверка вводимых символов и размера числа тоже верно 🙂

Но есть несколько проблем:

Предложение ввести число выводится 2 раза.

```
    mov cl,byte[bp-5]
```

Старшая часть регистра не обнуляется, это может вызвать ошибки. Лучше movzx.

```
    push bp
```

```
    sub bp,6
```

В середине процедуры это лишнее, я не понял зачем 🙂
Также pop bp после команды loop. В BP восстановится какой-то мусор из стека.

При обработке ошибки нельзя делать jmp start. В стеке остаётся буфер, сохраненное значение BP, адрес возврата.

[Ответить]

argir
05-01-2011 19:34

У меня получилась похожая программа, как выше.
Почему при операции compile не проходила команда

mov byte[bp-si],4

выдавая invalid address, хотя команды

mov byte[bp-6],4 и mov byte[bp+si],4 проходят?

[Ответить]

xrnd
11-01-2011 00:09

mov byte[bp-si],4

Нет такого режима адресации 🙂 Смотри часть 14.

Может быть только сложение значений.
bp-6 = bp+(-6) , Смещение равно -6.

[Ответить]

Гость
10-02-2011 15:05

```
use16
org 100h
mov [axz],5
call npu
call print_str
mov ax,4C00h
int 21h
;————————
dxa rw 1
axz rw 1
buffer rb 10
oshibka db 13,10,'Error!$',13,10
ok db 13,10,'OK!$',13,10
npu:
push sp ;————————————-Сохраняет как положено
push bp ;————————————-Сохраняет как положено
sub sp ,260;————- выделяем 260 байт в стёке с запасом
mov bp,sp
mov ax,[axz];———— Сколько символов можно вести
mov ah,0Ah ;————- номер функции для прерывания
mov [bp],ax;————- Помешаем в 1 байт Сколько символов можно вести
mov dx,bp;————— Адрес буфера для ведённых строк (стёк)
int 21h
mov cl,byte[bp+1];—- второй байт буфера = количеству ведённых символов
push cx;————————— сохраняем количества ведённых символов в стёк
mov si,2;————————- Так как символы со 2 байта начинаются
zl:
mov ah,byte[bp+si];— загружаем первый символ
sub ah ,30h ;0-9;—— превращаем строку в символ
cmp ah ,9;————— если больше 9
```

```
ja m1;————————- то ошибка
SAL ah,4;—————- сдвигаем было 0x0x стало x0x0
mov byte[bp+si],ah;— сохраняем изменения в стёк
inc si
loop zl
pop cx;———————— восстанавливаем счётчик
mov si,2;—————- Так как символы со 2 байта начинаются
zl1:
mov ah,byte[bp+si];— первый символ
mov al,byte[bp+si+1];- второй
SAR ah,4;—————- было x0x0 стало 0xx0
SAL ax,4;—————- было 0xx0 стало xx00
;si+1 = 00
mov [buffer+di],ah;— сохраняем в память число
inc di;————————
dec cx;————————
JCXZ m3;———————— если cx= 0 , закончились символы выход
add si,2
loop zl1
m3:
mov [dxa],ok
jmp m2
m1:
mov [dxa],oshibka ;
m2:
pop bp;———————————— А вот при восстановление слетают значения
pop sp;———————————— А вот при восстановление слетают значения
ret ;260
print_str:
mov ah,09h
mov dx,[dxa]
int 21h
```

Сама функция работает нормальна , до момента восстановления sp ,bp из стёка

А сами значение , берутся непонятно откуда ?
Не как понять не могу почему , там абсолютно не то , что должно.

[Ответить]

xrnd
10-02-2011 19:13

Значения слетают потому, что ты выполняешь команду «sub sp, 260»
А обратно не прибавляешь 260.

Ты в этой программе извратился по полной 😀 Но в десятичную систему число так и не преобразовал.
Ввести можно 4 символа, а должно быть только 3 (max «255»).
260 байт — это слишком большой запас. Для ввода нужно 3 байта + 1 байт CR и ещё 2 байта для длины буфера и строки. Всего получается 6 байтов.

Сохранять SP в стеке не нужно. Пролог лучше сделать примерно таким:

```
    push bp
    mov bp,sp
    sub sp,6
```

Посмотри ещё раз часть 23: http://asmworld.ru/uchebnyj-kurs/023-vvod-chisel-s-konsoli/

Всего-то нужно вместо глобального буфера для ввода строки сделать локальный.

[Ответить]

Гость
11-02-2011 03:36

Ясна , спасибо за разъяснения ,)
Почти преобразовал в 10 систему, в памяте 123 , а должна быть 3201.

Да я просто пытался понять чем стёк от обычный памяте отличается , в принцыпе как я понял не чем но у стёка есть + пара плюсов .
Это быстрая доступность необходимых данных.
Выделения и освобождения памяти.

[Ответить]

xrnd
11-02-2011 17:27

У тебя только сдвиги влево и вправо. А где же умножение на 10?

Стек хорошо подходит для временного хранения данных в процедуре.
Например, если вводится число с клавиатуры, то буфер со строкой нужен только внутри процедуры. После преобразования строки в число можно эту память освободить.

[Ответить]

Гость
11-02-2011 23:01

Да пожалуй , сдвигом не отделаться.
Десятичных чисел там просто нет , и быть не может.
Для отображения используется система с основание 16.
Зато если доработать , то с 16ричными числами можно обойтись без умножения.

Тут умножение не десятичных чисел будит а 16ричных , а это я и не учитывал ,)

P.S Спасибо теперь буду знать ,)

[Ответить]

Александр
10-08-2011 13:31

Скажите пожалуйста, зачем нужно корректировать регистр SP
sub sp,locals_size ;Выделение памяти для локальных переменных
а потом восстанавливать его значение
mov sp,bp ;Восстановление указателя стека
Он же (SP) не используется для адресации данных в стеке, а используется BP:
mov bp,sp
…
mov ax,[bp+4]
Или такой подход следованием инженерной этике? Т.е. чтобы читатель подпрограммы видел, сколько памяти в стеке будет использовано под локальные переменные? Ну так это можно написать в комментариях, а строка вида
sub sp,locals_size ;Выделение памяти для локальных переменных
все равно ничего не сообщает о распределении памяти стека, всего лишь о ее объеме, выделенном под локальные переменные.

[Ответить]

xrnd
17-09-2011 16:43

Стек ведь используется не только для локальных переменных, но ещё и для сохранения регистров и для вызова других процедур 🙂
Если не вычесть размер переменных из SP, то вызов

процедуры или прерывания DOS испортит локальные переменные.

[Ответить]

egochnpp
11-08-2011 10:50

Оставлял здесь комментарий, но он не отобразился — видимо, из-за того, что я не был зарегистрирован.

Вопрос такой. Для чего в прологе при создании локальных переменных корректируется регистр SP ?

mov bp,sp ;Копирование указателя стека в BP
sub sp,4 ;Выделение 4 байт для локальных переменных

Он же (SP) в дальнейшем не используется. Для адресации к локальным переменным используется BP:

mov word[bp-4],1234 ;Инициализация первой локальной переменной
mov byte[bp-2],10 ;Инициализация второй локальной переменной

В эпилоге SP восстанавливается:

mov sp,bp ;Восстановление указателя стека
pop bp ;Восстановление BP
ret 2 ;Возврат из процедуры

[Ответить]

xrnd
17-09-2011 16:45

Он же (SP) в дальнейшем не используется.
В том то и дело, что он используется 😉
Неявно — командами PUSH, POP, CALL, RET, INT.

[Ответить]

алекс
01-07-2012 04:38

```
;процедура ввода числа 0 — 225 (8 бит) с консоли.
;_____________________________________________
use16
org 100h
jmp start
erst1 db 'invalid symdol, try again (0…9)',13,10,'$'
erst2 db 'overflow, try again (0…225)',13,10,'$'
start:
call num_in
cmp ah,0
je outt
cmp ah,1
je erms1
cmp ah,2
je erms2
erms1:
mov ah,09h
mov dx,erst1
int 21h
jmp start
erms2:
mov ah,09h
mov dx,erst2
int 21h
jmp start
outt:
mov ax,4c00h
```

```
int 21h
;————————————процедура————————————
num_in:
push bp
mov bp,sp
sub sp,6
push dx
push cx
push si
mov byte[bp-6],4
mov dx,bp
sub dx,6
mov ah,0ah
int 21h
cmp byte[bp-5],0
je start
movzx cx,byte[bp-5]
mov si,-4
xor ax,ax
mov dh,10
conv:
mov dl,byte[bp+si]
inc si
cmp dl,’0′
jl error1
cmp dl,’9′
ja error1
sub dl,’0′
add al,dl
jc error2
cmp cx,1
je fin
mul dh
jc error2
loop conv
jmp fin
```

```
error1:
mov ah,1
jmp fin
error2:
mov ah,2
fin:
pop si
pop cx
pop dx
mov sp,bp
pop bp
ret
;_____________________________________________
```

[Ответить]

Андрей
03-06-2013 07:25

Боюсь что сильно опоздал со своим вопросом, но попробую. Вопрос касается примера процедуры с двумя локальными переменными.
В частности третей строки примера mov ax,[bp+4] ;AX = x
Не закралась ли тут опечатка? Ведь при использовании второй переменной (10) во всех действиях используется адресация [bp-2], как и при инициализации. А в третей строке почему то bp+4 плюс!?! Я чего то недопонял? Или это просто опечатка.

[Ответить]

Андрей
11-02-2014 02:48

Здравствуйте! Не нашёл куда написать. Помогите пожалуйста с fasm:
Написать программу, которая выводит на екран заранее

записаний ряд в таком порядке. У первом ряде 1-я буква с ряда, во втором ряде две буквы с ряда, в третьем ряде три буквы с ряда и т.д. и до последнего символа в ряде.

[Ответить]

Михаил
10-11-2015 15:02

Всё расписано хорошо и подробно, был бы лайк, то поставил бы.

[Ответить]

Олег
13-08-2019 14:02

Здравствуйте!
Написал процедуру для ввода десятичного числа (то 0 до 255), которая не выдаёт никаких ошибок, а просто не позволяет вводить некорректные данные. Можно вводить только допустимые цифры, удалять их бэкспейсом, а кнопкой enter завершать ввод, но при условии если введена хотя бы одна цифра. Процедура вроде работает правильно. Написал комментарии к коду.

```
;Procedure enter_byte
;выход:
; al — byte
enter_byte:
push bp
mov bp,sp
sub sp,4
push si
push dx
push ax
```

```
;====Вероятно необязательный участок кода====
mov byte[bp-1],’0′ ;обнуление локальных
mov byte[bp-2],’0′ ;переменных
mov byte[bp-3],’0′ ;символом «0»
;=================================================
mov si,-1 ;смещение относительно bp
;-1 — первая цифра
;-2 — вторая цифра
;-3 — третья цифра

;ввод символа без эха
;проверка символа
press_key_eb:
mov ah,8
int 21h
cmp al,8
jz key_backspace_eb
cmp al,13
jz key_enter_eb
cmp si,-4 ;если введены уже три цифры
jz press_key_eb
cmp si,-3 ;если введены две цифры
jz check_key_eb

;входит ли символ в диапазон цифр
continue_2_eb:
cmp al,’0′
jc press_key_eb
cmp al,’9′
ja press_key_eb

;отображение цифры на экране и запись её в локальную
переменную
continue_3_eb:
mov dl,al
mov ah,2
```

```
int 21h
mov [bp+si],al
dec si
jmp press_key_eb

;обработка нажатия backspace
key_backspace_eb:
cmp si,-1 ;ни одной цифры не введено
jz press_key_eb ;поэтому нечего удалять
;удаление символа с экрана———-
mov ah,2
mov dl,8 ;код backspace
int 21h
mov dl,32 ;код пробела
int 21h
mov dl,8
int 21h
;————————————————
inc si
;=========Скорее всего тоже не нужно==========
mov byte[bp+si],’0′ ;обнуление цифры в памяти
;=============================================
jmp press_key_eb

;обработка нажатия enter
key_enter_eb:
cmp si,-1 ;если не введено ни одной цифры
jz press_key_eb ;начинаем заново
cmp si,-2
jz one_digit_eb
cmp si,-3
jz two_digits_eb
jmp continue_1_eb ;если введено три цифры

;если введена одна цифра, то перемещаем её
;в конец, а первые две обнуляем символом «0»
```

```
one_digit_eb:
mov dh,byte[bp-1]
mov byte[bp-3],dh
mov byte[bp-1],30h
mov byte[bp-2],30h
jmp continue_1_eb

;если введены две цифры, то перемещаем их
;в конец, а первую обнуляем символом «0»
two_digits_eb:
mov dh,byte[bp-2]
mov byte[bp-3],dh
mov dh,byte[bp-1]
mov byte[bp-2],dh
mov byte[bp-1],30h
jmp continue_1_eb

;если введены две цифры
check_key_eb:
cmp byte[bp-1],’2′
ja press_key_eb ;первая больше 2, третью нельзя вводить
jc continue_2_eb ;первая меньше 2, то можно
;если первая равна 2
cmp byte[bp-2],’5′
ja press_key_eb ;вторая больше 5, третью нельзя вводить
jc continue_2_eb ;вторая меньше 5, то можно
;если вторая равна 5, то переопределяем диапазон
;для третьей
cmp al,’0′
jc press_key_eb
cmp al,’5′
ja press_key_eb
jmp continue_3_eb

;уменьшаем символы на 30h и преобразуем в байт без знака
continue_1_eb:
```

```
mov dh,100
mov al,[bp-1]
sub al,30h
mul dh
mov dl,al
mov dh,10
mov al,[bp-2]
sub al,30h
mul dh
add dl,al
mov al,[bp-3]
sub al,30h
add dl,al

pop ax
mov al,dl
pop dx
pop si
leave
ret
;EndProcedure
```

[Ответить]

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

Уведомить меня о новых комментариях по email.

Уведомлять меня о новых записях почтой.