Учебный курс. Часть 23. Ввод чисел с консоли

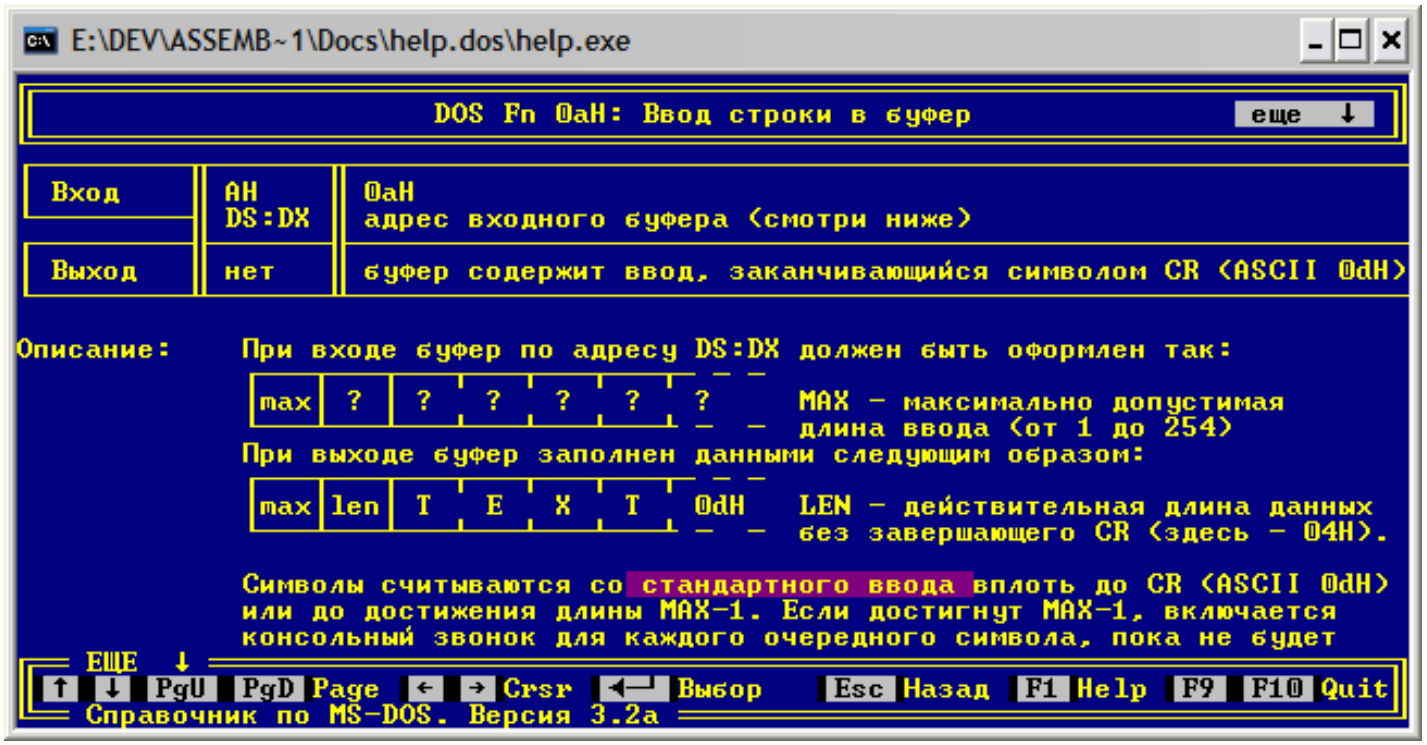
Автор: xrnd | Рубрика: Исходники, Учебный курс | 07-08-2010 |
Распечатать запись

В прошлой части мы научились преобразовывать числа в строку и выводить на консоль. А в этой займёмся обратной задачей — вводом чисел с консоли и преобразованием строки в число. Поскольку ввод в двоичном и восьмеричном виде используется редко, я рассмотрю только примеры ввода чисел в десятичном виде (со знаком и без знака) и в шестнадцатеричном.

Вводить числа сложнее, чем выводить, так как помимо преобразования необходимо проверять корректность введённой пользователем строки. Хорошая программа должна устойчиво работать при любых входных данных (в том числе специально введённых так, чтобы нарушить её работу).

Ввод строки с консоли

Для ввода строки можно использовать функцию MS-DOS 0Ah. Функция позволяет ввести строку длиной от 1 до 254 символов. При вызове в DX передаётся адрес буфера, первый байт которого должен содержать максимально допустимую длину строки. Длина считается вместе с символом конца строки CR (0dh). В результате работы функции во второй байт буфера записывается фактическая длина введённой строки (не считая символа CR). Начиная с третьего байта в буфер записываются символы строки. Подробнее о работе функции можно узнать в раритетном справочнике по DOS 🙂



Чтобы удобнее было использовать эту функцию, можно написать небольшую процедуру. Например, такую:

```
;Процедура ввода строки с консоли
;  вход: AL - максимальная длина (с символом CR) (1-254)
; выход: AL - длина введённой строки (не считая символа CR)
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
```

```
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
input_str:
    push cx                 ;Сохранение CX
    mov cx,ax               ;Сохранение AX в CX
    mov ah,0Ah              ;Функция DOS 0Ah - ввод строки в буфер
    mov [buffer],al         ;Запись максимальной длины в первый байт буфера
    mov byte[buffer+1],0    ;Обнуление второго байта (фактической длины)
    mov dx,buffer           ;DX = адрес буфера
    int 21h                 ;Обращение к функции DOS
    mov al,[buffer+1]       ;AL = длина введённой строки
    add dx,2                ;DX = адрес строки
    mov ah,ch               ;Восстановление AH

    pop cx                  ;Восстановление CX
    ret
...
buffer   rb 256
```

Процедура использует отдельно объявленный буфер. В качестве единственного параметра ей передаётся максимальная длина строки в регистре AL. После возврата из процедуры в этот регистр записывается фактическая длина строки, а в регистр DX — адрес начала строки. Старшая часть AX сохраняется.

Ввод десятичных чисел без знака

Для преобразования числа в строку используется так называемая схема Горнера. Любое число в десятичной системе можно представить в следующем виде:

$$347_{10} = 3 \cdot 10^2 + 4 \cdot 10^1 + 7 \cdot 10^0 = (3 \cdot 10 + 4) \cdot 10 + 7$$

Это очень удобно запрограммировать в цикле: результат сначала умножается на 10, а потом к нему прибавляется очередная цифра и так далее для всех цифр. Кстати, это верно не только для десятичной системы счисления, но и для других, разница только в множителе, который должен быть равен основанию системы счисления.

Следующая процедура преобразует строку в слово в регистре AX. Адрес строки передаётся в DX, длина строки передаётся в AL. Если строка не корректна, процедура возвращает 0 и устанавливает флаг CF. Ошибка возвращается в следующих случаях:

- строка имеет нулевую длину, то есть пустая строка;
- строка содержит любые символы кроме десятичных цифр;
- число находится вне границ диапазона представления чисел (для слова без знака 0… 65535).

```
;Процедура преобразования десятичной строки в слово без знака
;  вход: AL - длина строки
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
str_to_udec_word:
    push cx                 ;Сохранение всех используемых регистров
    push dx
    push bx
    push si
    push di

    mov si,dx               ;SI = адрес строки
    mov di,10               ;DI = множитель 10 (основание системы счисления)
    movzx cx,al             ;CX = счётчик цикла = длина строки
```

```
    jcxz studw_error        ;Если длина = 0, возвращаем ошибку
    xor ax,ax               ;AX = 0
    xor bx,bx               ;BX = 0

studw_lp:
    mov bl,[si]             ;Загрузка в BL очередного символа строки
    inc si                  ;Инкремент адреса
    cmp bl,'0'              ;Если код символа меньше кода '0'
    jl studw_error          ; возвращаем ошибку
    cmp bl,'9'              ;Если код символа больше кода '9'
    jg studw_error          ; возвращаем ошибку
    sub bl,'0'              ;Преобразование символа-цифры в число
    mul di                  ;AX = AX * 10
    jc studw_error          ;Если результат больше 16 бит - ошибка
    add ax,bx               ;Прибавляем цифру
    jc studw_error          ;Если переполнение - ошибка
    loop studw_lp           ;Команда цикла
    jmp studw_exit          ;Успешное завершение (здесь всегда CF = 0)

studw_error:
    xor ax,ax               ;AX = 0
    stc                     ;CF = 1 (Возвращаем ошибку)


studw_exit:
    pop di                  ;Восстановление регистров
    pop si
    pop bx
    pop dx
    pop cx
    ret
```

Для установки флага CF используется команда STC. Сбросить флаг CF можно командой CLC. В коде данной процедуры она не используется, так как в случае успешного завершения цикла флаг CF всегда будет равен 0.

На основе этой процедуры несложно написать ещё одну для ввода чисел размером 1 байт. Сначала строка преобразуется в слово без знака, а затем выполняется проверка старшей части на равенство нулю. Обратите внимание, что команда TEST всегда сбрасывает флаг CF.

```
;Процедура преобразования десятичной строки в байт без знака
;  вход: AL - длина строки
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AL - байт (в случае ошибки AL = 0)
;        CF = 1 - ошибка
str_to_udec_byte:
    push dx                 ;Сохранение регистров
    push ax
    call str_to_udec_word   ;Преобразование строки в слово (без знака)
    jc studb_exit           ;Если ошибка, то возвращаем ошибку
    test ah,ah              ;Проверка старшего байта AX
    jz studb_exit           ;Если 0, то выход из процедуры (здесь всегда CF = 0)
    xor al,al               ;AL = 0
    stc                     ;CF = 1 (Возвращаем ошибку)
studb_exit:
    pop dx
    mov ah,dh               ;Восстановление только старшей части AX
    pop dx
    ret
```

Следующие две процедуры совмещают ввод строки с преобразованием строки в число. Для слова нужно ввести максимум 5 символов, а для байта — максимум 3.

```
;Процедура ввода слова с консоли в десятичном виде (без знака)
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
input_udec_word:
    push dx                 ;Сохранение DX
    mov al,6                ;Ввод максимум 5 символов (65535) + конец строки
    call input_str          ;Вызов процедуры ввода строки
    call str_to_udec_word   ;Преобразование строки в слово (без знака)
    pop dx                  ;Восстановление DX
    ret
```

```
;Процедура ввода байта с консоли в десятичном виде (без знака)
; выход: AL - байт (в случае ошибки AL = 0)
;        CF = 1 - ошибка
input_udec_byte:
    push dx                 ;Сохранение DX
    mov al,4                ;Ввод максимум 3 символов (255) + конец строки
    call input_str          ;Вызов процедуры ввода строки
    call str_to_udec_byte   ;Преобразование строки в байт (без знака)
    pop dx                  ;Восстановление DX
    ret
```

Ввод десятичных чисел со знаком

Ввод чисел со знаком немного труднее. Необходимо проверить первый символ строки: если это символ ‘-‘, то число отрицательное. Кроме того, нужно внимательно проверить диапазон представления (для слова со знаком -32768…32767). Строку без символа ‘-‘ можно преобразовать процедурой для беззнакового числа.

```
;Процедура преобразования десятичной строки в слово со знаком
;  вход: AL - длина строки
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
str_to_sdec_word:
    push bx                 ;Сохранение регистров
    push dx

    test al,al              ;Проверка длины строки
    jz stsdw_error          ;Если равно 0, возвращаем ошибку
    mov bx,dx               ;BX = адрес строки
    mov bl,[bx]             ;BL = первый символ строки
    cmp bl,'-'              ;Сравнение первого символа с '-'
    jne stsdw_no_sign       ;Если не равно, то преобразуем как число без знака
    inc dx                  ;Инкремент адреса строки
    dec al                  ;Декремент длины строки
stsdw_no_sign:
    call str_to_udec_word   ;Преобразуем строку в слово без знака
    jc stsdw_exit           ;Если ошибка, то возвращаем ошибку
    cmp bl,'-'              ;Снова проверяем знак
    jne stsdw_plus          ;Если первый символ не '-', то число положительное
    cmp ax,32768            ;Модуль отрицательного числа должен быть не больше 32768
    ja stsdw_error          ;Если больше (без знака), возвращаем ошибку
    neg ax                  ;Инвертируем число
    jmp stsdw_ok            ;Переход к нормальному завершению процедуры
stsdw_plus:
    cmp ax,32767            ;Положительное число должно быть не больше 32767
```

```
    cmp ax,32767            ;Положительное число должно быть не больше 32767
    ja stsdw_error          ;Если больше (без знака), возвращаем ошибку

stsdw_ok:
    clc                     ;CF = 0
    jmp stsdw_exit          ;Переход к выходу из процедуры
stsdw_error:
    xor ax,ax               ;AX = 0
    stc                     ;CF = 1 (Возвращаем ошибку
stsdw_exit:
    pop dx                  ;Восстановление регистров
    pop bx
    ret
```

Обратите внимание, что для перехода после проверки диапазона используется команда JA (как для сравнения чисел без знака). Ввод байта со знаком реализуется с помощью той же процедуры и дополнительной проверки диапазона значения.

```
;Процедура преобразования десятичной строки в байт со знаком
;  вход: AL - длина строки
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AL - байт (в случае ошибки AL = 0)
;        CF = 1 - ошибка
str_to_sdec_byte:
    push dx                 ;Сохранение регистров
    push ax
    call str_to_sdec_word   ;Преобразование строки в слово (со знаком)
    jc stsdb_exit           ;Если ошибка, то возвращаем ошибку
    cmp ax,127              ;Сравнение результата с 127
    jg stsdb_error          ;Если больше - ошибка
    cmp ax,-128             ;Сравнение результата с -128
    jl stsdb_error          ;Если меньше - ошибка
    clc                     ;CF = 0
    jmp studb_exit          ;Переход к выходу из процедуры
stsdb_error:
    xor al,al               ;AL = 0
    stc                     ;CF = 1 (Возвращаем ошибку)
stsdb_exit:
    pop dx
    mov ah,dh               ;Восстановление только старшей части AX
    pop dx
    ret
```

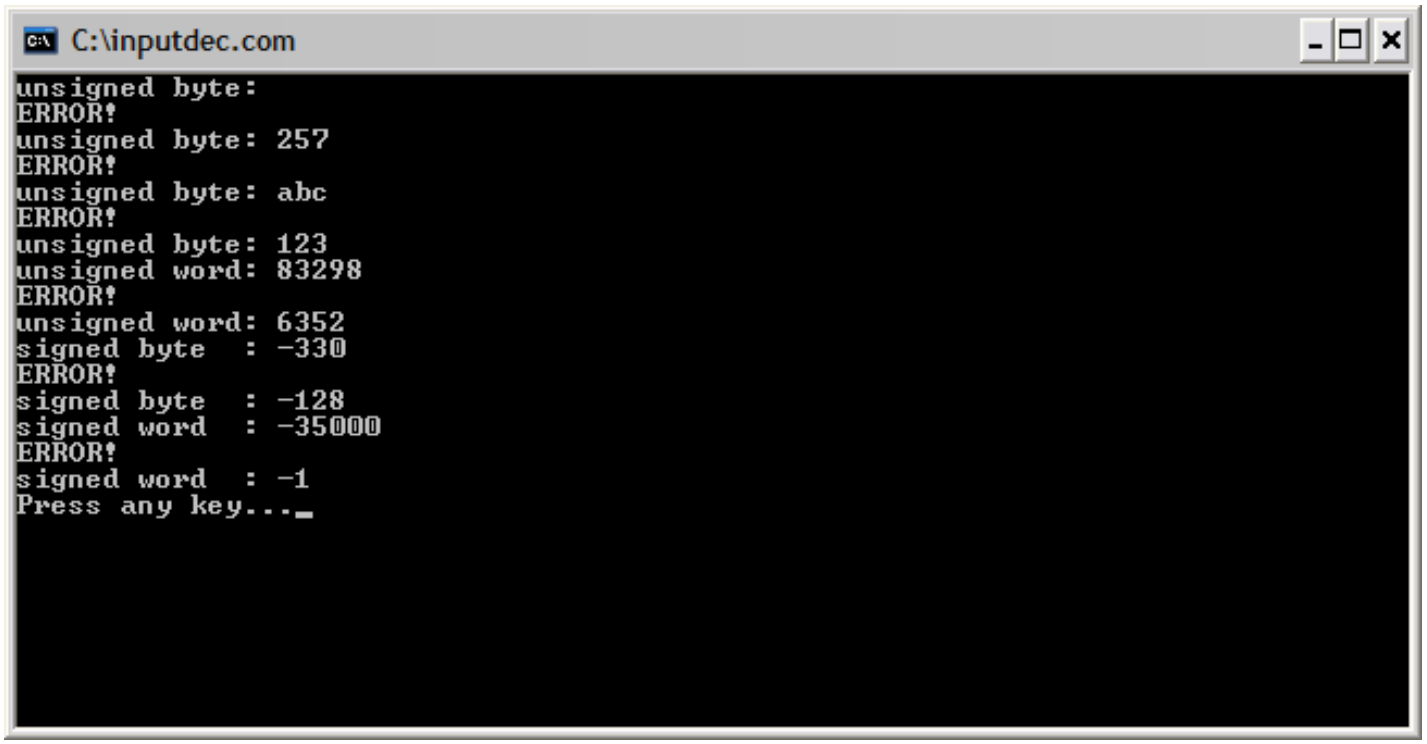
Наконец, ещё две процедуры совмещают ввод строки с преобразованием её в слово и байт со знаком.

```
;Процедура ввода слова с консоли в десятичном виде (со знаком)
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
input_sdec_word:
    push dx                 ;Сохранение DX
    mov al,7                ;Ввод максимум 7 символов (-32768) + конец строки
    call input_str          ;Вызов процедуры ввода строки
    call str_to_sdec_word   ;Преобразование строки в слово (со знаком)
    pop dx                  ;Восстановление DX
    ret
```

```
;Процедура ввода байта с консоли в десятичном виде (со знаком)
; выход: AL - байт (в случае ошибки AL = 0)
;        CF = 1 - ошибка
input_sdec_byte:
```

```
input_sdec_byte:
    push dx                 ;Сохранение DX
    mov al,5                ;Ввод максимум 3 символов (-128) + конец строки
    call input_str          ;Вызов процедуры ввода строки
    call str_to_sdec_byte   ;Преобразование строки в байт (со знаком)
    pop dx                  ;Восстановление DX
    ret
```

Полный исходный код примера вы можете скачать отсюда: inputdec.asm. В случае некорректной строки программа выводит сообщение об ошибке и повторяет запрос ввода числа:



Ввод шестнадцатеричных чисел

Преобразование шестнадцатеричной строки в число несколько проще. Удобно реализовать в виде отдельной процедуры преобразование одной цифры. Процедура воспринимает символы ‘A’-‘F’ независимо от регистра. Так как перед вычитанием выполняются проверки, флаг CF всегда будет равен нулю после успешного преобразования.

```
;Процедура преобразования шестнадцатеричной цифры в число
;  вход: DL - символ-цифра
; выход: DL - значение цифры (0-15, в случае ошибки DL = 0)
;        CF = 1 - ошибка
convert_hex_digit:
    cmp dl,'0'              ;Сравнение с символом '0'
    jl chd_error            ;Если меньше, возвращаем ошибку
    cmp dl,'9'              ;Сравнение с символом '9'
    jg chd_a_f              ;Если больше, то возможно это буква a-f или A-F
    sub dl,'0'              ;Преобразование цифры в число
    ret                     ;Возврат из процедуры (здесь всегда CF = 0)

chd_a_f:
    and dl,11011111b        ;Преобразование буквы в верхний регистр
    cmp dl,'A'              ;Сравнение с символом 'A'
    jl chd_error            ;Если меньше, возвращаем ошибку
    cmp dl,'F'              ;Сравнение с символом 'F'
    jg chd_error            ;Если больше, возвращаем ошибку
    sub dl,'A'-10           ;Преобразуем букву в число
```

```
    ret                     ;Возврат из процедуры (здесь тоже всегда CF = 0)

chd_error:
    xor dl,dl               ;DL = 0
    stc                     ;CF = 1
    ret                     ;Возврат из процедуры
```

Теперь легко можно написать преобразование шестнадцатеричной строки в слово. Вместо умножения на 16 в процедуре используется сдвиг на 4 бита влево, а вместо сложения — операция ИЛИ. Проверки диапазона значения не нужны, достаточно проверить длину строки и преобразовать цифры.

```
;Процедура преобразования шестнадцатеричной строки в слово
;  вход: AL - длина строки
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
str_to_hex_word:
    push cx                 ;Сохранение регистров
    push dx
    push si

    movzx cx,al             ;CX = счётчик цикла = длина строки
    jcxz sthw_error         ;Если длина строки = 0, возвращаем ошибку
    cmp cx,4
    jg sthw_error           ;Если длина строки больше 4, возвращаем ошибку
    xor ax,ax               ;AX = 0
    mov si,dx               ;SI = адрес строки

sthw_lp:
    mov dl,[si]             ;Загрузка в DL очередного символа строки
    inc si                  ;Инкремент адреса строки
    call convert_hex_digit  ;Преобразование шестнадцатеричной цифры в число
    jc sthw_error           ;Если ошибка, то возвращаем ошибку
    shl ax,4                ;Сдвиг AX на 4 бита влево
    or al,dl                ;Добавление преобразованной цифры
    loop sthw_lp            ;Команда цикла
    jmp sthw_exit           ;CF = 0

sthw_error:
    xor ax,ax               ;AX = 0
    stc                     ;CF = 1

sthw_exit:
    pop si                  ;Восстановление регистров
    pop dx
    pop cx
    ret
```

Для ввода байта используется та же процедура, но дополнительно проверяется длина строки — она должна быть не больше 2.

```
;Процедура преобразования шестнадцатеричной строки в байт
;  вход: AL - длина строки
;        DX - адрес строки, заканчивающейся символом CR(0Dh)
; выход: AL - байт (в случае ошибки AL = 0)
;        CF = 1 - ошибка
str_to_hex_byte:
    push cx                 ;Сохранение CX
    mov cx,ax               ;Сохранение AX в CX
    cmp al,2                ;Проверка длины строки
```

```
    jg sthb_error           ;Если больше 2, возвращаем ошибку
    call str_to_hex_word    ;Преобразование строки в слово
    jnc sthb_exit           ;Если нет ошибки, то переход к выходу из процедуры
sthb_error:
    stc                     ;CF = 1
sthb_exit:
    mov ah,ch               ;Восстановление AH
    pop cx                  ;Восстановление CX
    ret
```

Ещё две процедуры для ввода и преобразования строки, также как для десятичного ввода:

```
;Процедура ввода слова с консоли в шестнадцатеричном виде
; выход: AX - слово (в случае ошибки AX = 0)
;        CF = 1 - ошибка
input_hex_word:
    push dx                 ;Сохранение DX
    mov al,5                ;Ввод максимум 4 символов (FFFF) + конец строки
    call input_str          ;Вызов процедуры ввода строки
    call str_to_hex_word    ;Преобразование строки в слово
    pop dx                  ;Восстановление DX
    ret
```

```
;Процедура ввода байта с консоли в шестнадцатеричном виде
; выход: AL - байт (в случае ошибки AL = 0)
;        CF = 1 - ошибка
input_hex_byte:
    push dx                 ;Сохранение DX
    mov al,3                ;Ввод максимум 2 символов (FF) + конец строки
    call input_str          ;Вызов процедуры ввода строки
    call str_to_hex_byte    ;Преобразование строки в байт
    pop dx                  ;Восстановление DX
    ret
```

Полный исходный код примера: inputhex.asm. Как и в примере с десятичными числами, программа повторяет запрос ввода, пока не будут введены корректные данные:

Упражнение

Напишите программу для ввода байта с консоли в двоичном виде. Желательно с проверкой корректности ввода 🙂 Результаты можете писать в комментариях или на форуме.

Ещё раз ссылки на примеры:

- inputdec.asm — ввод десятичных чисел с консоли (со знаком и без)
- inputhex.asm — ввод шестнадцатеричных чисел с консоли

Следующая часть »

Комментарии:

fufel
08-09-2010 19:07

Здаравсвуйте!
Вот, кажется работает.

```
use16
org 100h
jmp start
;————————————-

vvod db 13,10,'Vvedite byte v dvoichnom vide: $'
pak db 13,10,'Press any key…$'
oshibka db 13,10,'Error!$',13,10
ok db 13,10,'OK!$',13,10
buffer rb 10
vved_byte rb 1
;————————————
start:
mov byte[buffer],10
mov byte[buffer+1],0
mov dx,vvod
mov ah,09h
int 21h
mov dx,buffer
mov ah,0Ah
int 21h
add dx,2
mov al,byte[buffer+1]
cmp al,8
jne error
xor di,di
mov di,2
mov cx,8
lp:
mov dl,byte[buffer+di]
cmp dl,'0′
jne odin
shl bl,1
```

inc di
loop lp
jmp fsjo
odin:
cmp dl,’1′
jne error
or bl,00000001b
shl bl,1
inc di
loop lp
fsjo:
mov [vved_byte],bl
mov dx,ok
mov ah,09h
int 21h
mov dx,pak
mov ah,09h
int 21h
mov ah,08h
int 21h
mov ax,4c00h
int 21h

error:
mov dx,oshibka
mov ah,09h
int 21h
jmp start

[Ответить]

xrnd
09-09-2010 23:01

Привет!
Хорошая программа, работает правильно, но есть ошибки.

Можно ввести 9 символов. При этом размер буфера оказывается мал. Последний символ вводится в vved_byte, а символ конца строки затирает первый байт команды mov byte[buffer],10. По счастливой случайности, твоей программе это не мешает работать 🙂

Тебе нужно в первый байт буфера записать 9, а не 10.

```
    mov [buffer],9
```

Тогда размер буфера 9+2 = 11 байтов

```
buffer rb 11
```

Кроме того, можно твою программу сильно оптимизировать. Убрать xor di,di, убрать add dx,2 (похоже вообще бесполезное действие) и не делить цикл на 2 ветки. У тебя получается 2 команды loop и ещё по две повторяющихся команды.

[Ответить]

fufel
13-09-2010 19:28

А-а-а, теперь понял чего она глючила. Исправил, спасибо.

[Ответить]

argir
28-12-2010 10:16

Привет!
При решении заданий иногда хочется повторить какой-нибудь предыдущий урок.
Было бы удобней, если была бы ссылка «предыдущая часть».

[Ответить]

xrnd
29-12-2010 22:05

Хорошая идея. Сделаю обязательно 🙂

[Ответить]

argir
28-12-2010 16:58

Вот, что у меня получилось:

```
; ввод байта с консоли в двоичном виде
use16
org 100h
jmp start
lb db 11 dup(9) ;1(MAX)+1(LEN)+8(TEXT)+1(0DH)=11
;запись мах длины,чтоб не пусто было 🙂 (8+1(CR))
lb1 db ?
s_byte db 'Enter byte: $'
s_error db 13,10,'ERROR!',13,10,'$'
s_pak db 13,10,'OK! Press any key for exit',13,10,'$'
start:
mov dx,s_byte
call print_str; печать 'Enter byte: '
call input_str; ввод чего-нибудь
call str_to_byte; преобразование строкового байта
jnc in_end ;;если не было ошибок — выход
mov dx,s_error
call print_str ;печать 'ERROR!'
jmp start; может что-нибудь введут правильное
in_end:
mov dx,s_pak
call print_str ;печать 'Press any key…'
```

```
mov ah,8
int 21h

mov ax,4C00h
int 21h
;_____________________________________________
;Процедура ввода строки с консоли
;выход CX = длина введённой строки и
;в DX = адрес строки
input_str:
push ax
mov dx,lb ;DX = адрес буфера
mov ah,0Ah ;
int 21h ;ввод строки
movzx cx,[lb+1] ;CX = длина введённой строки
add dx,2
pop ax
ret
;_____________________________________________
Процедура преобразования
;в lb1 результат
str_to_byte:
push cx
push bx
push si

mov si,dx ;SI = DX
xor ax,ax ;AX = 0
xor bx,bx ;BX = 0

studw_lp:
jcxz error ;Если длина введённой строки = 0, возвращаем ошибку
lp1: mov bl,[si] ;BL=символ строки
cmp bl,'0′ ;символ= '0'?
jz stu ;если да, то преобразуем
cmp bl,'1′ ;символ='1′?
jnz error ; если нет, возвращаем ошибку
stu: sub bl,'0′ ;\
shl al,1 ; преобразуем
add al,bl ;/
inc si ;
loop lp1 ;
jmp studw_exit ;завершаем

error:
xor ax,ax ;AX = 0
stc ;CF = 1 (ERROR)

studw_exit:
mov [lb1],al;сохраняем полученное
pop si
pop bx
pop cx
ret
```

;————————————————————————————————
;Процедура вывода строки на консоль
; DX — адрес строки
print_str:
push ax
mov ah,9
int 21h
pop ax
ret

[Ответить]

xrnd
29-12-2010 22:27

Получилось хорошо, всё правильно написано.

Можно немного сократить проверку введённого символа. Вместо первой команды сравнения сразу делать вычитание символа ‘0’. Команда CMP делает то же самое, но результат не сохраняется.

```
lp1: mov bl,[si] ;BL=символ строки
    sub bl,'0' ;символ= '0'?
    jz stu ;если да, то преобразуем
    cmp bl,1 ;символ='1'?
    jnz error ; если нет, возвращаем ошибку
stu: shl al,1 ; преобразуем
    add al,bl
```

[Ответить]

Гость
30-01-2011 20:13

use16
org 100h
mov al,9;———максимальный размер 8(9-1)
call input_str; —-процедура сохраняет ведённые значения
xor dx,dx ;—очищаем так как процедура изменила значение
xor bx,bx ;
xor ax,ax ;
mov al,11111110b ; этот байт используется для обнуление 1 бита
mov bh,11111111b; этот байт значение
mov si,2 ; — данные начинаются со смещения 2
mov cl,[buffer+1] ; узнаём длину , ведёной строки
cmp cx,8 ; если длинна больше 8 или меньше ошибка
jnz metca3

zicol:
JCXZ metca4 ; может возникнуть ситуация при которой CX=0 , a loop -1 и cx= ffff
mov bl,byte[buffer+si] ; выгружаем значения из памяти в bl
inc si
sub bl ,30h ; 30h -для десятичных чисел , чтобы преобразовать строку в число
jz metca2 ; если вели 0 — (30-30) FZ=0

```
cmp bl ,1 ; если вели 1 — (1-1) FZ=0
jz metca1 ;1
jnz metca3; если вели не 1 и не 0 (ошибка), сообщение об ошибка будет выведена ;на экран
loop zicol

jmp metca4 ; если ошибок не было программа закроется

metca1: ; если вели 1 , цыкал просто уменьшится на 1
dec cx
jmp zicol

metca2:;если вели 0
push cx ; сохраняем cx
push ax ;для того что бы ноль всегда был 0(битом) и не менял положение при ;сдвиги
dec cx ; тут уменьшается счётчик сдвига на 1 , так как ноль это 0 бит
; если сдвинуть на 1 , обнулится 2 бит а не первый
rol al,cl ; сдвик
AND bh,al ; копируем результат в bh
pop ax ; восстанавливаем позицию нуля в байте
pop cx
dec cx ; уменьшаем счётчик цыкал
jmp zicol
metca3:
mov dx ,error
mov ah,09h
int 21h
mov ah,08h
int 21h
metca4:
mov ax,4C00h
int 21h
;_____
error db 'error$'
buffer rb 256
rezultat rb 2

input_str: ; позаимствована у автора ,)
push cx
mov cx,ax
mov ah,0Ah
mov [buffer],al
mov byte[buffer+1],0
mov dx,buffer
int 21h
mov al,[buffer+1]
add dx,2
mov ah,ch
pop cx
ret
```

[Ответить]

xrnd
09-02-2011 14:31

Вроде работает, но есть ошибки.

Обнулять регистры не обязательно, если ты всё равно что-то в них записываешь.
Кстати, позаимствованная процедура возвращает адрес строки и длину, можно не вычислять заново.

Вот здесь сравнение может быть неверным:

```
    mov cl,[buffer+1] ; узнаём длину , ведёной строки
    cmp cx,8 ; если длинна больше 8 или меньше ошибка
```

Длина записывается только в младшую часть CX, поэтому и сравнивать надо CL, либо явно обнулить CH.

```
    jz metca1 ;1
    jnz metca3; если вели не 1 и не 0 (ошибка)
    loop zicol
    jmp metca4 ; если ошибок не было программа закроется
```

В этом коде команда LOOP никогда не будет выполнена. Переход, если ноль, и переход, если не ноль — то есть в любом случае будет переход 🙂 Команда JMP тоже лишняя.

Ты используешь интересный алгоритм — если символ ‘0’, то сбрасывается соответствующий бит (если я правильно понял).

Следующий код можно оптимизировать:

```
metca1: ; если вели 1 , цыкал просто уменьшится на 1
    dec cx
    jmp zicol

metca2:;если вели 0
    push cx ; сохраняем cx
    push ax ;для того что бы ноль всегда был 0(битом)
    dec cx ; тут уменьшается счётчик сдвига на 1 , так как ноль это 0 бит
    ; если сдвинуть на 1 , обнулится 2 бит а не первый
    rol al,cl ; сдвик
    AND bh,al ; копируем результат в bh
    pop ax ; восстанавливаем позицию нуля в байте
    pop cx
    dec cx ; уменьшаем счётчик цыкал
    jmp zicol
```

Например, разместить метку metca1 перед двумя последними командами.
Или сохранять CX не обязательно, если после извлечения из стека снова выполняется декремент. Сохранять и восстанавливать AX тоже как-то сложно.
Можно просто присваивать AL нужное значение.

```
metca1: ; если вели 1 , цыкал просто уменьшится на 1
    dec cx
    jmp zicol

metca2:;если вели 0
    mov al,0FEh
    dec cx
    rol al,cl ; сдвик
```

```
    AND bh,al ; копируем результат в bh
    jmp zicol
```

[Ответить]

Гость
12-02-2011 00:38

Стало гораздо проще .
Спасибо ,что тратещ своё время на разбегание такова зачистившую бреда ,)
Кроме этого сайта я негде и не практиковался в ассемблере .

[Ответить]

xrnd
14-02-2011 12:43

Для начинающего у тебя неплохо получается. Просто некоторые вещи приходят с опытом.

[Ответить]

feanarmo
14-10-2011 11:52

Столкнулся с проблемой написании программы. При вычислении результирующего
байта в цикле на loopz происходит выход при CX > 0 и любом значении ZF.
Версия компилятора fasm 1.69
Версия отладчика td 3.0

; Напишите программу для ввода байта с консоли в двоичном виде.
; Желательно с проверкой корректности ввода.

```
use16
org 100h
jmp START
;_____________________________
hex_digit db '0123456789ABCDEF'
out_str db 13,10,' h',13,10,'$'
err_str db 13,10,'Error binary digit',13,10,'$'
;_____________________________
START:
; Получаем двоичное представление байта в виде строки
mov al, 8
call input_str

; Строка корректна, если не содержит символов кроме 0 и 1
movzx cx, al
mov di, 0FFFFh
LOOP1:
inc di
cmp byte[bx+di], '0'
jz END_LOOP1
cmp byte[bx+di], '1'
```

```
END_LOOP1:
loopz LOOP1 ; ??? При CX > 0 и любом значении ZF цикл завершается

jz END_IF1
mov ah, 09h
mov dx, err_str
int 21h
jmp EXIT
END_IF1:

; Преобразуем строку в байт AH
movzx si, al ; Вычисляем смещение для последнего символа
mov cx, si
dec si
xor ah, ah
LOOP2:
mov dl, [bx+si]
sub dl, '0'
or ah, dl
shl ah, 1
dec si
loop LOOP2

; Выводим строку с hex представлением AH
mov al, ah ; AL — 0-3 биты
; AH — 4-7 биты
and al, 00001111b
shr ah, 4

movzx si, ah ; Заносим старший разряд
mov bl, [hex_digit+si]
mov [out_str+2], bl

movzx si, al ; Заносим младший разряд
mov bl, [hex_digit+si]
mov [out_str+3], bl

mov ah, 09h
mov dx, out_str
int 21h
EXIT:
mov ax, 4C00h
int 21h
;————————————————————
;————————————————————
input_str: ; Процедура ввода с клавиатуры
; AL — максимальный размер строки
; Результат:
; AL — длина введенной строки (не считая CR)
; BX — адрес строки, с CR на конце (0Dh)
push dx
push cx
```

```
mov cl, ah
mov ah,0Ah ; Функция DOS 0Ah — ввод строки в буфер
mov [str_buf],al ; Запись максимальной длины в первый байт буфера
mov byte[str_buf+1],0 ; Обнуление второго байта (фактической длины)
mov dx, str_buf ; DX = адрес буфера
int 21h ; Обращение к функции DOS
mov al,[str_buf+1] ; AL = длина введённой строки
mov bx,dx
add bx,2 ; DX = адрес строки

mov ah, cl
pop cx
pop dx
ret
;———————————————
str_buf rb 256
```

[Ответить]

feanarmo
14-10-2011 15:17

Сам дурак — сам виноват. При отладке по F8 проскачил данный цикл и не проверил di.

[Ответить]

Олег
16-10-2011 11:21

А как быть с двойным словом?

[Ответить]

xrnd
27-10-2011 00:50

Ввод проще сделать. примерно также как ввод слова.
Только сложение надо делать по частям, в два этапа)))
Опять же, если нужно — я готов написать пример кода.

[Ответить]

VlaDi4eKK
19-11-2013 14:10

Если вам не сложно, то напишите. А то столкнулся с проблемой, что не могу вывести строку после ввода, выводятся крокозябры…=(

[Ответить]

MALYSh
12-12-2011 00:16

Автор, спасибо тебе огромное. Спас, (в буквальном смысле), мою задницу от незачета по программированию.

[Ответить]

алекс
29-03-2012 20:39

```
use16
org 100h
jmp start
buffer rb 11
bbyte db 0
erm db 13,10,'Invalid symbol, try again…',13,10,'$'
prim db 'Hello, enter your binary byte',13,10,'$'
start:
mov ah,09h
mov dx,prim
int 21h
cont:
mov dx,buffer
mov ah,0ah
mov byte[buffer],9
int 21h
movzx cx,byte[buffer+1]
mov si,2

loop_bin:
movzx bx,[buffer+si]
inc si
cmp bx,'0′
jz next
cmp bx,'1′
jnz error
stc
next:
rcl [bbyte],1
loop loop_bin

jmp term
error:
mov ah,09h
mov dx,erm
int 21h
jmp cont
term:
mov ax,4c00h
int 21h
```

[Ответить]

Михаил
10-01-2017 17:40

```
sub bl,'0′ ;Преобразование символа-цифры в число
```

вот этот момент не как не освещен, не понятно что ASCII символа-цифры отнять от строки ‘0’ получится цифра. Или я не правильно понял?

[Ответить]

Олег
14-07-2019 22:19

Здравствуйте!
Решил написать универсальную функцию, которая преобразует строку цифр любой длины в целое положительное число указанного размера. Как ею пользоваться: перед вызовом функции необходимо в регистр BP положить адрес строки, в которой находятся цифры, а в регистр BX — адрес числа, в CX необходимо указать размер числа в байтах. Также функция выдаёт код ошибки в регистр DL. Например, если цифр слишком много, а в CX указано мало байтов, то старшая часть числа (старшие байты) отбросятся, и в регистре DL запишется код ошибки равный 2. Если будет 0, то всё нормально.
Как функция работает. Десятичное число делится на 256 по частям, точнее по одной цифре, после деления остаток записывается в первый байт числа. Далее на 256 делится то что осталось от предыдущего деления, а остаток записывается в следующий байт числа. Так происходит до тех пор, пока не заполнятся все байты числа, размер которого указан в регистре CX. Изначально я хотел делить на 16. Но в остатке будет полубайт, младший или старший, а это как-то замороченно. Я решил, что проще делить на 256. Но вот проблема, 256 не помещается в 8-битный регистр. Тогда я подумал, что можно использовать команду сдвига регистра вправо. Число 256 же степень двойки. Можно сдвигать регистр на 8 разрядов. Но это деление без остатка, а мне и результат нужен и остаток. И тут мне пришла гениальная мысль — а зачем собственно делить? Ведь когда мы делим 16-битное число на 256, то результатом является содержимое старшего байта (старшего регистра), а остатком — содержимое младшего байта (младшего регистра). То есть, делить ничего не надо. Фактически я делил без команды деления, используя умножение и сложение. Вот же ж как бывает.
Функция не оптимизирована на скорость, поэтому может выполняться много лишних действий. Не хотел больше голову ломать. С другой стороны код более короткий и более понятный. Проверял всяко, вроде работает нормально, но кто его знает.

```
;Function convert_string_to_integer
;Вход
;bp — адрес строки. На конце $
;bx — адрес числа
;cx — размер числа в байтах
;Выход
;dl — код ошибки
; 0 — ок
; 1 — строка пустая
; 2 — переполнение, для старших байтов в числе не хватило места
; 3 — указан нулевой размер числа в регистре CX
; 4 — указан нулевой адрес строки в регистре BP
; 5 — указан нулевой адрес числа в регистре BX
; 6 — в строке содержатся недопустимые символы
convert_string_to_integer:
push ax
push bx
push cx
```

```
push si
push di

;проверка входных данных
cmp cx,0
jz error_3_func_csti
cmp bp,0
jz error_4_func_csti
cmp bx,0
jz error_5_func_csti

;проверка строки на пустоту
xor si,si
cmp byte[bp+si],'$'
jz error_1_func_csti

;проверка строки на символы
loop_6_func_csti:
cmp byte[bp+si],'$'
jz continue_1_func_csti
cmp byte[bp+si],29h
jna error_6_func_csti
cmp byte[bp+si],39h
ja error_6_func_csti
inc si
cmp si,1000 ;на всякий случай, вдруг отсутствует $
jz error_6_func_csti
jmp loop_6_func_csti

;сохранение строки в стек, и понижение символов на 30h
continue_1_func_csti:
xor si,si
loop_1_func_csti:
mov al,[bp+si]
push ax
sub al,30h
mov [bp+si],al
inc si
cmp byte[bp+si],'$'
jnz loop_1_func_csti

;преобразование строки в число
mov dl,10
loop_3_func_csti:
xor di,di
xor ax,ax
loop_2_func_csti:
mul dl
add al,[bp+di]
adc ah,0
mov [bp+di],ah
inc di
cmp di,si
jnz loop_2_func_csti
```

```
mov [bx],al
inc bx
loop loop_3_func_csti

;проверка на переполнение
xor di,di
loop_4_func_csti:
cmp byte[bp+di],0
jnz error_2_func_csti
inc di
cmp di,si
jnz loop_4_func_csti
xor dl,dl

;восстановление строки из стека
loop_5_func_csti:
dec si
pop ax
mov [bp+si],al
cmp si,0
jnz loop_5_func_csti
jmp exit_func_csti

;запись в DL кода ошибки
error_1_func_csti:
mov dl,1
jmp exit_func_csti
error_2_func_csti:
mov dl,2
jmp loop_5_func_csti
error_3_func_csti:
mov dl,3
jmp exit_func_csti
error_4_func_csti:
mov dl,4
jmp exit_func_csti
error_5_func_csti:
mov dl,5
jmp exit_func_csti
error_6_func_csti:
mov dl,6
jmp exit_func_csti

exit_func_csti:
pop di
pop si
pop cx
pop bx
pop ax
ret
;EndFunction
```

[Ответить]

Олег
16-07-2019 00:07

Дополнил функцию возможностью преобразовывать строку цифр в отрицательное число.
Строка должна начинаться с символа «-»

```
;Function convert_string_to_integer
;Вход
;bp — адрес строки. На конце $
;bx — адрес числа
;cx — размер числа в байтах
;Выход
;dl — код ошибки
; 0 — ок
; 1 — строка пустая
; 2 — переполнение, не хватает места для записи числа.
; Результат некорректный
; 3 — указан нулевой размер числа в регистре CX
; 4 — указан нулевой адрес строки в регистре BP
; 5 — указан нулевой адрес числа в регистре BX
; 6 — в строке содержатся недопустимые символы
; 7 — переполнение для отрицательного числа.
; Отброшен один старший байт числа со значением FF.
; В редких случаях FF на конце не обязателен, например,
; положительное 128 равно -128, если число имеет размер 1 байт,
; но ошибка 7 будет установлена
convert_string_to_integer:
push ax
push si
push di
push bp
push bx
push cx

;проверка входных данных
cmp cx,0
jz error_3_func_csti
cmp bp,0
jz error_4_func_csti
cmp bx,0
jz error_5_func_csti

;проверка строки на пустоту и на знак вначале
xor si,si
cmp byte[bp+si],'+'
jz continue_2_func_csti
cmp byte[bp+si],'-'
jnz continue_3_func_csti
continue_2_func_csti:
inc bp
continue_3_func_csti:
cmp byte[bp+si],'$'
jz error_1_func_csti
```

```
;проверка строки на символы
loop_6_func_csti:
cmp byte[bp+si],'$'
jz continue_1_func_csti
cmp byte[bp+si],29h
jna error_6_func_csti
cmp byte[bp+si],39h
ja error_6_func_csti
inc si
cmp si,1000 ;на всякий случай, вдруг отсутствует $
jz error_6_func_csti
jmp loop_6_func_csti

;сохранение строки в стек, и понижение символов на 30h
continue_1_func_csti:
xor si,si
loop_1_func_csti:
mov al,[bp+si]
push ax
sub al,30h
mov [bp+si],al
inc si
cmp byte[bp+si],'$'
jnz loop_1_func_csti

;преобразование строки в число
mov dl,10
loop_3_func_csti:
xor di,di
xor ax,ax
loop_2_func_csti:
mul dl
add al,[bp+di]
adc ah,0
mov [bp+di],ah
inc di
cmp di,si
jnz loop_2_func_csti
mov [bx],al
inc bx
loop loop_3_func_csti

;проверка на переполнение
xor di,di
loop_4_func_csti:
cmp byte[bp+di],0
jnz error_2_func_csti
inc di
cmp di,si
jnz loop_4_func_csti
xor dl,dl
```

```
;восстановление строки из стека
loop_5_func_csti:
dec si
pop ax
mov [bp+si],al
cmp si,0
jnz loop_5_func_csti
cmp dl,0
jnz exit_func_csti

;преобразование положительного числа в отрицательное
pop cx
pop bx
pop bp
push bp
push bx
push cx
xor si,si
cmp byte[bp+si],'-'
jnz exit_func_csti
mov di,cx
dec di
test byte[bx+di],10000000b
jnz error_7_func_csti
loop_7_func_csti:
not byte[bx+si]
inc si
cmp si,di
jna loop_7_func_csti
xor si,si
dec cx
add byte[bx+si],1
loop_8_func_csti:
inc si
adc byte[bx+si],0
loop loop_8_func_csti
jmp exit_func_csti

;запись в DL кода ошибки
error_1_func_csti:
mov dl,1
jmp exit_func_csti
error_2_func_csti:
mov dl,2
jmp loop_5_func_csti
error_3_func_csti:
mov dl,3
jmp exit_func_csti
error_4_func_csti:
mov dl,4
jmp exit_func_csti
error_5_func_csti:
mov dl,5
```

```
jmp exit_func_csti
error_6_func_csti:
mov dl,6
jmp exit_func_csti
error_7_func_csti:
mov dl,7
jmp loop_7_func_csti

exit_func_csti:
pop cx
pop bx
pop bp
pop di
pop si
pop ax
ret
;EndFunction
```

[Ответить]

Олег
08-08-2019 21:35

Написал ещё пару функций и программу, которая их использует. Суть программы: ввести строку цифр в консоли со знаком или без, затем строка преобразуется в число, а после число обратно преобразуется в строку и выводится в консоль, где их можно сверить и проверить правильность работы подпрограмм. Размеры строк и числа можно ставить любые, главное чтобы хватило памяти, поэтому в разумных пределах. Функцию для ввода числа в консоли решил написать свою. Она отличается от досовской тем, что можно вводить только цифры и знак «-» вначале, не имеет ограничения на размер (главное чтобы хватило памяти), в конце строки ставится символ «$», завершение происходит при нажатии enter или введено максимум символов. Недостаток моей функции — нельзя удалять введённые символы, в досовской это можно делать, но работает некорректно при переходе на следующую строку, поэтому такую возможность решил не добавлять. Может можно сделать, пока не знаю.

```
use16
org 100h

jmp start
;var
stroka_input rb 50 ;размер переменных можно менять
stroka_output rb 100 ;но так же необходимо поменять их
chislo rb 30 ;в основной программе, где есть комментарии
new_line db 13,10,'$'
msg_input db 'Enter chislo: $'
msg_output db 'You enter: $'

start:
mov ah,9
mov dx,new_line
int 21h
mov dx,msg_input
int 21h
```

```
mov bx,stroka_input
mov cx,50 ;размер переменной stroka_input в байтах
call input_string_of_integer
mov dx,new_line
int 21h
mov bp,stroka_input
mov bx,chislo
mov cx,30 ;размер числа в байтах
call convert_string_to_integer
mov bp,chislo
mov si,30 ;размер числа в байтах
mov bx,stroka_output
mov di,100 ;размер переменной stroka_output в байтах
call convert_integer_to_string

mov dx,msg_output
int 21h
mov dx,stroka_output
int 21h
mov dx,new_line
int 21h

mov ax,4c00h
int 21h

;Function input_string_of_integer
;Вход
;bx — адрес строки
;cx — длина строки в байтах вместе с $
;Выход
;dl — код ошибки
; 0 — ok
; 1 — нулевой адрес строки
; 2 — длина строки меньше трёх символов
input_string_of_integer:
push ax
push bx
push cx

;проверка входных данных
cmp bx,0
jz error_1_func_isoi
cmp cx,3
jc error_2_func_isoi

dec cx

;ввод хотя бы одной цифры или минуса
loop_3_func_isoi:
mov ah,8
int 21h
cmp al,'-'
jz continue_1_func_isoi
cmp al,30h
```

```
jc loop_3_func_isoi
cmp al,39h
ja loop_3_func_isoi
jmp continue_2_func_isoi

;отображение и запись минуса
continue_1_func_isoi:
mov ah,2
mov dl,al
int 21h
mov [bx],al
inc bx
dec cx

;если «-«, то ввод хотя бы одной цифры
loop_2_func_isoi:
mov ah,8
int 21h
cmp al,30h
jc loop_2_func_isoi
cmp al,39h
ja loop_2_func_isoi
jmp continue_2_func_isoi

;ввод цифр, отображение и запись в строку
loop_1_func_isoi:
mov ah,8
int 21h
cmp al,13
jz noerror_func_isoi
cmp al,30h
jc loop_1_func_isoi
cmp al,39h
ja loop_1_func_isoi
continue_2_func_isoi:
mov ah,2
mov dl,al
int 21h
mov [bx],al
inc bx
loop loop_1_func_isoi

;запись в dl кода ошибки
noerror_func_isoi:
mov byte[bx],'$'
mov dl,0
jmp exit_func_isoi

error_1_func_isoi:
mov dl,1
jmp exit_func_isoi

error_2_func_isoi:
mov dl,2
```

```
exit_func_isoi:
pop cx
pop bx
pop ax
ret
;EndFunction

;Function convert_string_to_integer
;Вход
;bp — адрес строки. На конце $
;bx — адрес числа
;cx — размер числа в байтах
;Выход
;dl — код ошибки
; 0 — ок
; 1 — строка пустая
; 2 — переполнение, не хватает места для записи числа.
; Результат некорректный
; 3 — указан нулевой размер числа в регистре CX
; 4 — указан нулевой адрес строки в регистре BP
; 5 — указан нулевой адрес числа в регистре BX
; 6 — в строке содержатся недопустимые символы
; 7 — переполнение для отрицательного числа.
; Отброшен один старший байт числа со значением FF.
; В редких случаях FF на конце не обязателен, например,
; положительное 128 равно -128, если число имеет размер 1 байт,
; но ошибка 7 будет установлена
convert_string_to_integer:
push ax
push si
push di
push bp
push bx
push cx

;проверка входных данных
cmp cx,0
jz error_3_func_csti
cmp bp,0
jz error_4_func_csti
cmp bx,0
jz error_5_func_csti

;проверка строки на пустоту и на знак вначале
xor si,si
cmp byte[bp+si],'+'
jz continue_2_func_csti
cmp byte[bp+si],'-'
jnz continue_3_func_csti
continue_2_func_csti:
inc bp
continue_3_func_csti:
```

```
cmp byte[bp+si],'$'
jz error_1_func_csti

;проверка строки на символы
loop_6_func_csti:
cmp byte[bp+si],'$'
jz continue_1_func_csti
cmp byte[bp+si],29h
jna error_6_func_csti
cmp byte[bp+si],39h
ja error_6_func_csti
inc si
cmp si,1000 ;на всякий случай, вдруг отсутствует $
jz error_6_func_csti
jmp loop_6_func_csti

;сохранение строки в стек, и понижение символов на 30h
continue_1_func_csti:
xor si,si
loop_1_func_csti:
mov al,[bp+si]
push ax
sub al,30h
mov [bp+si],al
inc si
cmp byte[bp+si],'$'
jnz loop_1_func_csti

;преобразование строки в число
mov dl,10
loop_3_func_csti:
xor di,di
xor ax,ax
loop_2_func_csti:
mul dl
add al,[bp+di]
adc ah,0
mov [bp+di],ah
inc di
cmp di,si
jnz loop_2_func_csti
mov [bx],al
inc bx
loop loop_3_func_csti

;проверка на переполнение
xor di,di
loop_4_func_csti:
cmp byte[bp+di],0
jnz error_2_func_csti
inc di
cmp di,si
```

```
jnz loop_4_func_csti
xor dl,dl

;восстановление строки из стека
loop_5_func_csti:
dec si
pop ax
mov [bp+si],al
cmp si,0
jnz loop_5_func_csti
cmp dl,0
jnz exit_func_csti

;преобразование положительного числа в отрицательное
pop cx
pop bx
pop bp
push bp
push bx
push cx
xor si,si
cmp byte[bp+si],'-'
jnz exit_func_csti
mov di,cx
dec di
test byte[bx+di],10000000b
jnz error_7_func_csti
loop_7_func_csti:
not byte[bx+si]
inc si
cmp si,di
jna loop_7_func_csti
xor si,si
dec cx
add byte[bx+si],1
loop_8_func_csti:
inc si
adc byte[bx+si],0
loop loop_8_func_csti
jmp exit_func_csti

;запись в DL кода ошибки
error_1_func_csti:
mov dl,1
jmp exit_func_csti
error_2_func_csti:
mov dl,2
jmp loop_5_func_csti
error_3_func_csti:
mov dl,3
jmp exit_func_csti
error_4_func_csti:
mov dl,4
```

```
jmp exit_func_csti
error_5_func_csti:
mov dl,5
jmp exit_func_csti
error_6_func_csti:
mov dl,6
jmp exit_func_csti
error_7_func_csti:
mov dl,7
jmp loop_7_func_csti

exit_func_csti:
pop cx
pop bx
pop bp
pop di
pop si
pop ax
ret
;EndFunction

;Function convert_integer_to_string
;Вход
;bp — адрес числа
;si — размер числа в байтах
;bx — адрес строки
;di — размер строки, включая $
;Выход
;dx — код ошибки
; ffff — нулевой адрес числа в bp
; fffe — нулевой размер числа в si
; fffd — нулевой адрес строки в bx
; fffc — указан размер строки в di меньше 3 символов
; 0 и более — количество цифр, которые
; не поместились в конце строки.
; 0 означает, что все цифры поместились.
convert_integer_to_string:

push ax
push bx
push cx
push si
push di

;проверка входных данных
cmp bp,0
jz error_ffff_func_cits
cmp si,0
jz error_fffe_func_cits
cmp bx,0
jz error_fffd_func_cits
cmp di,3
jc error_fffc_func_cits
```

```
;сохраняем число в стек
dec si
xor ax,ax
xor cx,cx
loop_1_func_cits:
mov al,[bp+si]
push ax
inc cx
dec si
jns loop_1_func_cits
push cx

;проверка знака числа
;отрицательное число преобразуется в положительное
mov si,cx
dec si
test byte[bp+si],10000000b
jz continue_1_func_cits
mov byte[bx],'-'
inc bx
dec di
loop_2_func_cits:
not byte[bp+si]
dec si
jns loop_2_func_cits
dec cx
xor si,si
add byte[bp+si],1
loop_3_func_cits:
inc si
adc byte[bp+si],0
loop loop_3_func_cits

;преобразование числа в строку
;запись цифр в стек
continue_1_func_cits:
inc si
mov dl,10
xor cx,cx

loop_4_func_cits:
dec si
js continue_2_func_cits
cmp byte[bp+si],0
jz loop_4_func_cits
push si
xor ax,ax
loop_5_func_cits:
mov al,[bp+si]
div dl
mov [bp+si],al
dec si
jns loop_5_func_cits
```

```
pop si
inc si
push ax
inc cx
jmp loop_4_func_cits
continue_2_func_cits:
dec di
xor si,si
cmp cx,0
jnz loop_6_func_cits
mov byte[bx],’0′
inc bx
mov byte[bx],’$’
xor dx,dx
jmp continue_4_func_cits

;запись цифр из стека в строку
loop_6_func_cits:
pop ax
add ah,30h
mov byte[bx+si],ah
inc si
dec di
jz error_lack_func_cits
loop loop_6_func_cits
xor dx,dx
continue_3_func_cits:
mov byte[bx+si],’$’

;восстановление числа из стека
continue_4_func_cits:
pop cx
xor si,si
loop_7_func_cits:
pop ax
mov byte[bp+si],al
inc si
loop loop_7_func_cits
jmp exit_func_cits

;запись кода ошибки в dx
error_ffff_func_cits:
mov dx,0ffffh
jmp exit_func_cits
error_fffe_func_cits:
mov dx,0fffeh
jmp exit_func_cits
error_fffd_func_cits:
mov dx,0fffdh
jmp exit_func_cits
error_fffc_func_cits:
mov dx,0fffch
jmp exit_func_cits
```

```
;недостаточно места для записи последних цифр.
;в dx указывается сколько цифр отброшено
error_lack_func_cits:
dec cx
mov dx,cx
loop_8_func_cits:
pop ax
loop loop_8_func_cits
jmp continue_3_func_cits

exit_func_cits:
pop di
pop si
pop cx
pop bx
pop ax
ret
;EndFunction
```

[Ответить]

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

Уведомить меня о новых комментариях по email.

Уведомлять меня о новых записях почтой.