

Учебный курс. Часть 22. Вывод чисел на консоль

Автор: xrnd | Рубрика: [Исходники](#), [Учебный курс](#) | 31-07-2010 | 

[Распечатать запись](#)

В качестве примера программирования процедур займёмся такой важной проблемой, как вывод на консоль чисел в различных системах счисления. Проблема эта возникает потому, что в ассемблере нет никаких специальных средств для вывода чисел, а с помощью стандартных функций можно выводить только строки.

Следовательно, задача сводится к тому, чтобы преобразовать двоичное число в строку символов, а затем вывести эту строку на экран. Все процедуры в этой части являются лишь примерами, вы можете использовать их или написать свои собственные процедуры, более удобные для вас.

Для начала рассмотрим две полезные процедуры, которые будут использоваться в дальнейшем. Чтобы постоянно не обращаться в коде в функции DOS 09h, удобно написать маленькую процедуру для вывода строки:

```
;Процедура вывода строки на консоль
; DI - адрес строки
print_str:
    push ax
    mov ah,9                ;Функция DOS 09h - вывод строки
    xchg dx,di              ;Обмен значениями DX и DI
    int 21h                 ;Обращение к функции DOS
    xchg dx,di              ;Обмен значениями DX и DI
    pop ax
    ret
```

В качестве параметра ей передаётся адрес строки в регистре DI. Строка должна оканчиваться символом '\$'. Здесь используется команда [XCHG](#), которая выполняет обмен значениями двух операндов.

Вторая полезная процедура — вывод конца строки. Она вызывает первую процедуру для вывода двух символов CR(13) и LF(10). Вызывается без параметров и не изменяет регистры.

```
;Процедура вывода конца строки (CR+LF)
print_endline:
    push di
    mov di,endlne           ;DI = адрес строки с символами CR,LF
    call print_str          ;Вывод строки на консоль
    pop di
    ret
...
endlne db 13,10,'$'
```

Вывод чисел в двоичном виде

Алгоритм вывода в двоичном виде очень прост. Нужно проанализировать все биты числа и поместить в строку символы '0' или '1' в зависимости от значения соответствующего бита. Удобно делать это с помощью циклического сдвига в цикле. Сдвинутый бит оказывается в флаге CF, а после завершения цикла в регистре то же значение, что и в начале. Процедура

byte_to_bin_str преобразует байт в регистре AL в строку. Адрес буфера для строки передаётся в регистре DI. Процедура всегда записывает в буфер 8 символов, так как в байте 8 бит.

```
;Процедура преобразования байта в строку в двоичном виде
; AL - байт.
; DI - буфер для строки (8 символов). Значение регистра не сохраняется.
byte_to_bin_str:
    push cx                ;Сохранение CX
    mov cx,8              ;Счётчик цикла

btbs_lp:
    rol al,1              ;Циклический сдвиг AL влево на 1 бит
    jc btbs_1             ;Если выдвинутый бит = 1, то переход
    mov byte[di], '0'     ;Добавление символа '0' в строку
    jmp btbs_end
btbs_1:
    mov byte[di], '1'     ;Добавление символа '1' в строку
btbs_end:
    inc di                ;Инкремент DI
    loop btbs_lp          ;Команда цикла

    pop cx                ;Восстановление CX
    ret                   ;Возврат из процедуры
```

Используя эту процедуру, легко написать ещё одну для вывода слова в двоичном виде:

```
;Процедура преобразования слова в строку в двоичном виде
; AX - слово
; DI - буфер для строки (16 символов). Значение регистра не сохраняется.
word_to_bin_str:
    xchg ah,al            ;Обмен AH и AL
    call byte_to_bin_str  ;Преобразование старшего байта в строку
    xchg ah,al            ;Обмен AH и AL
    call byte_to_bin_str  ;Преобразование младшего байта в строку
    ret
```

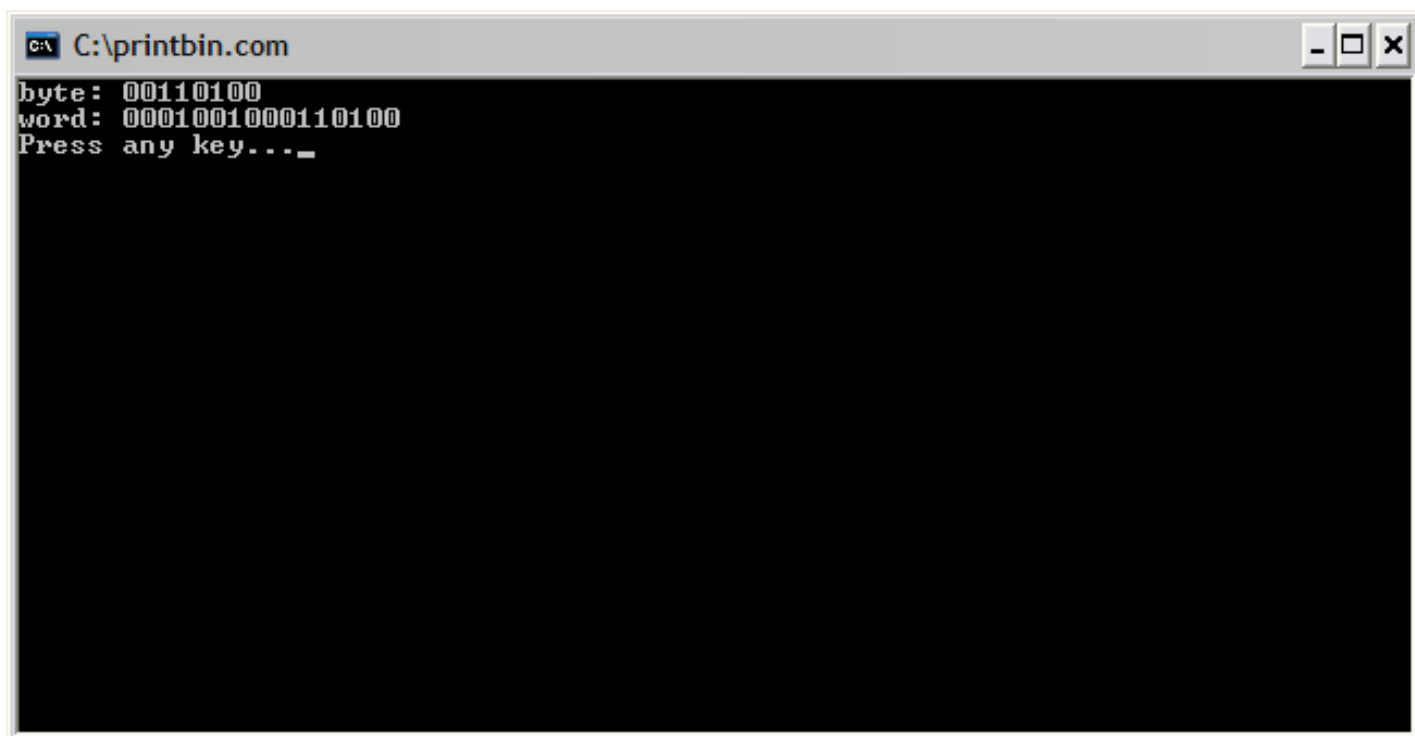
И наконец вот две процедуры, которые делают то, что нужно 😊 Буфер имеет размер 17 символов, так как в слове 16 бит + символ '\$', обозначающий конец строки.

```
;Процедура вывода байта на консоль в двоичном виде
; AL - байт
print_byte_bin:
    push di
    mov di,buffer         ;DI = адрес буфера
    call byte_to_bin_str  ;Преобразование байта в AL в строку
    mov byte[di], '$'     ;Добавление символа конца строки
    sub di,8              ;DI = адрес начала строки
    call print_str        ;Вывод строки на консоль
    pop di
    ret

;Процедура вывода слова на консоль в двоичном виде
; AX - слово
print_word_bin:
    push di
    mov di,buffer         ;DI = адрес буфера
    call word_to_bin_str  ;Преобразование слова в AX в строку
    mov byte[di], '$'     ;Добавление символа конца строки
    sub di,16             ;DI = адрес начала строки
    call print_str        ;Вывод строки на консоль
```

```
pop di
ret
...
buffer rb 17
```

Полный исходный код примера вы можете скачать отсюда: printbin.asm. Результат работы программы выглядит вот так:



Вывод чисел в шестнадцатеричном виде

Этот пример по структуре похож на предыдущий, поэтому для краткости я рассмотрю только сами процедуры преобразования числа в строку. Преобразование в шестнадцатеричный вид удобно выполнять группами по 4 бита, то есть по тетрадам. Каждая тетрада будет представлять собой одну шестнадцатеричную цифру. Я написал отдельную процедуру для преобразования тетрады в символ цифры:

```
;Процедура преобразования числа (0-15) в шестнадцатеричную цифру
; вход : AL - число (0-15)
; выход: AL - шестнадцатеричная цифра ('0'-'F')
to_hex_digit:
    add al,'0'           ;Прибавляем символ '0' (код 0x30)
    cmp al,'9'           ;Сравнение с символом '9' (код 0x39)
    jle thd_end          ;Если получилось '0'-'9', то выход
    add al,7             ;Прибавляем ещё 7 для символов 'A'-'F'
thd_end:
    ret
```

Если значение тетрады от 0 до 9, то достаточно только прибавить код символа '0' (0x30). А если значение больше 9, то надо прибавить ещё 7, чтобы получилась буква 'A'-'F'.

Теперь легко можно преобразовать байт в шестнадцатеричную строку, достаточно каждую из его тетрад заменить соответствующей цифрой:

```
;Процедура преобразования байта в строку в шестнадцатеричном виде
; AL - байт.
; DI - буфер для строки (2 символа). Значение регистра не сохраняется.
```

```

byte_to_hex_str:
    push ax
    mov ah,al           ;Сохранение значения AL в AH
    shr al,4           ;Выделение старшей тетрады
    call to_hex_digit   ;Преобразование в шестнадцатеричную цифру
    mov [di],al         ;Добавление символа в строку
    inc di              ;Инкремент DI
    mov al,ah           ;Восстановление AL
    and al,0Fh          ;Выделение младшей тетрады
    call to_hex_digit   ;Преобразование в шестнадцатеричную цифру
    mov [di],al         ;Добавление символа в строку
    inc di              ;Инкремент DI
    pop ax
    ret

```

Преобразование слова также не представляет трудности — сначала преобразуем старший байт, затем младший:

```

;Процедура преобразования слова в строку в шестнадцатеричном виде
; AX - слово
; DI - буфер для строки (4 символа). Значение регистра не сохраняется.
word_to_hex_str:
    xchg ah,al          ;Обмен AH и AL
    call byte_to_hex_str ;Преобразование старшего байта в строку
    xchg ah,al          ;Обмен AH и AL
    call byte_to_hex_str ;Преобразование младшего байта в строку
    ret

```

Полный исходный код примера: [printhex.asm](#). Результат работы программы выглядит вот так:



```

C:\printhex.com
byte: 5F
word: A15F
Press any key...

```

Вывод чисел в десятичном виде

С десятичными числами немного сложнее. Для начала займёмся числами без знака. Чтобы преобразовать число в десятичную строку необходимо в цикле делить его на 10 (это основание системы счисления). Остатки от деления дают нам значения десятичных цифр.

Первый остаток — младшая цифра, последний — старшая. Деление продолжается пока частное не равно нулю.

Например, если есть число 125. Делим его на десять: получаем 12, 5 в остатке. Потом делим 12 на десять: получаем 1, 2 в остатке. Наконец, 1 делим на 10: получаем 0, 1 в остатке. Цифры числа, начиная с младшей: 5, 2, 1. Так как обычно десятичные числа пишут, начиная со старшей цифры, то необходимо переставить их наоборот 😊 Я для этого использовал стек.

В первом цикле производится деление, полученные остатки преобразуются в цифры и помещаются в стек. Во втором цикле символы извлекаются из стека (в обратном порядке) и помещаются в строку. Так как максимальное значение слова без знака 65536 (5 цифр), то в буфер записывается максимум 5 символов.

```
;Процедура преобразования слова в строку в десятичном виде (без знака)
; AX - слово
; DI - буфер для строки (5 символов). Значение регистра не сохраняется.
word_to_uddec_str:
    push ax
    push cx
    push dx
    push bx
    xor cx,cx                ;Обнуление CX
    mov bx,10                ;В BX делитель (10 для десятичной системы)

wtuds_lp1:                  ;Цикл получения остатков от деления
    xor dx,dx                ;Обнуление старшей части двойного слова
    div bx                   ;Деление AX=(DX:AX)/BX, остаток в DX
    add dl,'0'               ;Преобразование остатка в код символа
    push dx                  ;Сохранение в стеке
    inc cx                   ;Увеличение счетчика символов
    test ax,ax               ;Проверка AX
    jnz wtuds_lp1            ;Переход к началу цикла, если частное не 0.

wtuds_lp2:                  ;Цикл извлечения символов из стека
    pop dx                   ;Восстановление символа из стека
    mov [di],dl              ;Сохранение символа в буфере
    inc di                   ;Инкремент адреса буфера
    loop wtuds_lp2           ;Команда цикла

    pop bx
    pop dx
    pop cx
    pop ax
    ret
```

Для вывода байта можно преобразовать его в слово и воспользоваться той же процедурой:

```
;Процедура преобразования байта в строку в десятичном виде (без знака)
; AL - байт.
; DI - буфер для строки (3 символа). Значение регистра не сохраняется.
byte_to_uddec_str:
    push ax
    xor ah,ah                ;Преобразование байта в слово (без знака)
    call word_to_uddec_str    ;Вызов процедуры для слова без знака
    pop ax
    ret
```

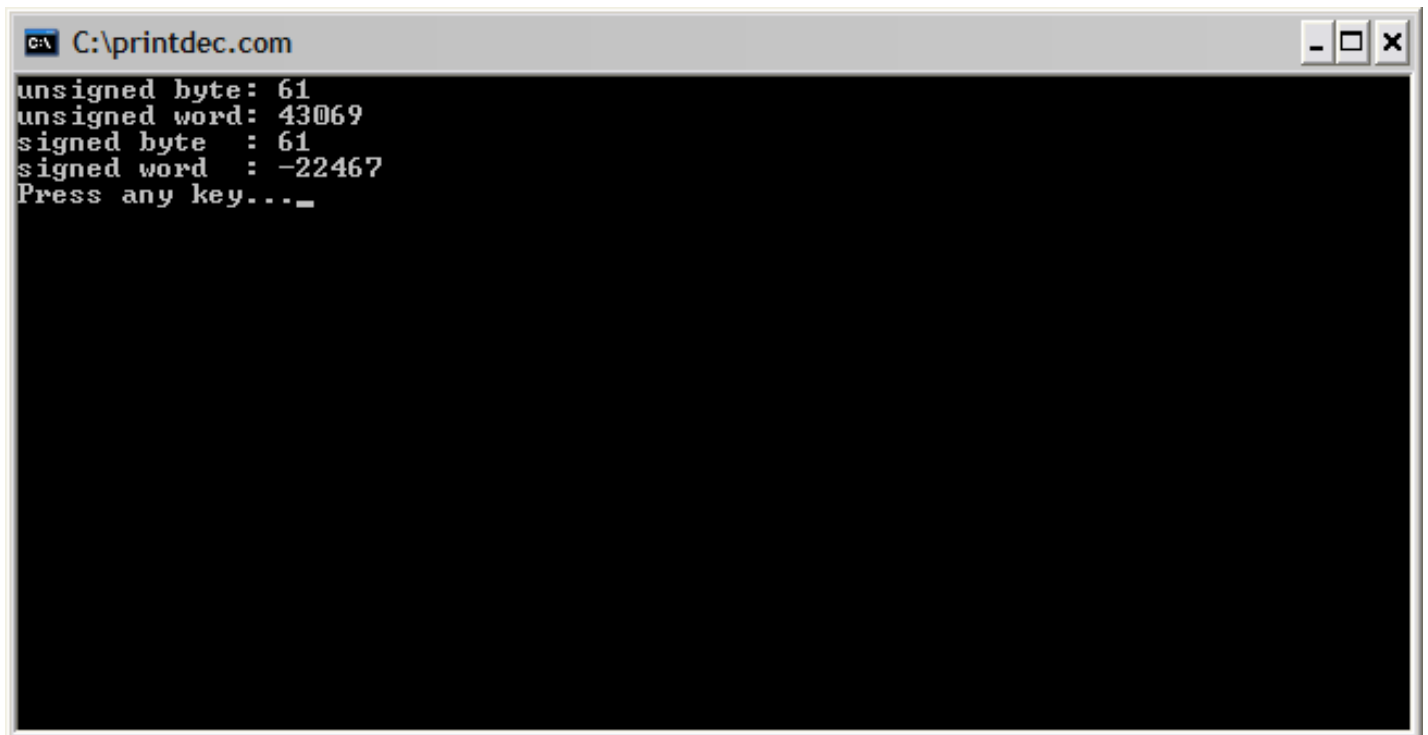
Теперь разберёмся с числами со знаком. Сначала нужно проверить старший бит числа. Если число положительное, то его можно преобразовать также как число без знака. Если число

отрицательное, то добавляем в строку символ '-', а затем инвертируем число и преобразуем как беззнаковое.

```
;Процедура преобразования слова в строку в десятичном виде (со знаком)  
; AX - слово  
; DI - буфер для строки (6 символов). Значение регистра не сохраняется.  
word_to_sdec_str:  
    push ax  
    test ax,ax           ;Проверка знака AX  
    jns wtsds_no_sign    ;Если >= 0, преобразуем как беззнаковое  
    mov byte[di], '-'    ;Добавление знака в начало строки  
    inc di               ;Инкремент DI  
    neg ax              ;Изменение знака значения AX  
wtsds_no_sign:  
    call word_to_udec_str ;Преобразование беззнакового значения  
    pop ax  
    ret
```

```
;Процедура преобразования байта в строку в десятичном виде (со знаком)  
; AL - байт.  
; DI - буфер для строки (4 символа). Значение регистра не сохраняется.  
byte_to_sdec_str:  
    push ax  
    movsx ax,al          ;Преобразование байта в слово (со знаком)  
    call word_to_sdec_str ;Вызов процедуры для слова со знаком  
    pop ax  
    ret
```

Полный исходный код примера: [printdec.asm](#). Результат работы программы выглядит вот так:



Как вывести на консоль в десятичном виде очень большое число (> 32 бит) читайте [здесь](#).

Вывод чисел в восьмеричном виде

Выводить числа в восьмеричном виде приходится достаточно редко, поэтому подробно описывать не буду. Можно либо делить число последовательно на 8, либо преобразовывать в

цифры группы по 3 бита. Я использовал второй вариант. Смотрите код примера: [printoct.asm](#).
Результат работы программы:



```
C:\printoct.com
byte: 253
word: 104653
Press any key...
```

Вывод чисел в других системах счисления

Реализуется также, как вывод в десятичном виде — с помощью алгоритма последовательного деления на основание системы счисления. Например, если вам нужно вывести число в пятеричной системе счисления, делить надо на 5, а не на 10.

Упражнение

Напишите программу для вывода на консоль массива слов со знаком в десятичном виде (например, через запятую). Для вывода чисел можете воспользоваться моим примером или написать свою собственную процедуру. Результаты можете писать в комментариях или на форуме.

Ещё раз ссылки на все примеры

- [printbin.asm](#) — вывод чисел на консоль в двоичном виде
- [printoct.asm](#) — вывод чисел на консоль в восьмеричном виде
- [printdec.asm](#) — вывод чисел на консоль в десятичном виде (со знаком и без знака)
- [printheX.asm](#) — вывод чисел на консоль в шестнадцатеричном виде

[Следующая часть »](#)

Комментарии:

fufel
28-08-2010 17:51

Здравствуйте!
use16
org 100h

```

jmp start
;-----
n db 5
m db 3
table:
dw 35,-43,7,98,-75
dw 65,-56,76,23,96
dw -32,48,-64,54,21
vozvr_kar db 13,10,»$»
buffer rb 5
;-----
start:
mov bx,10
movzx cx,[m]
xor si,si
lp_1:
push cx
movzx cx,[n]
lp_2:
mov ax,[table+si]
call des_vid
add si,2
cmp cx,1
je end_str
call zpt
loop lp_2

end_str:
call vozvr_kar
pop cx
loop lp_1

mov ah,08h
int 21h
mov ax,4c00h
int 21h

des_vid:
push cx
push dx
xor di,di
xor dx,dx
xor cx,cx
test ax,ax
jns net_mnsa
call minus
neg ax
net_mnsa:
cmp ax,10
jl fsjo
div bx
call dlshe
jmp net_mnsa

```



```
dlshe:
add dl,'0'
mov byte[buffer+di],dl
inc di
inc cx
ret
fsjo:
mov dl,al
call dlshe
dec di
vivod_smv1:
mov ah,02h
movzx dx,[buffer+di]
int 21h
dec di
loop vivod_smv1
pop dx
pop cx
ret
```

```
zpt:
push ax
push dx
mov ah,02h
mov dl,', '
int 21h
pop dx
pop ax
ret
```

```
minus:
push dx
push ax
mov dl,'-'
mov ah,02h
int 21h
pop ax
pop dx
ret
```

```
vozvr_kar:
push dx
push ax
mov ah,09h
mov dx,vozwr_kar
int 21h
pop ax
pop dx
ret
```

Вот только не понял, когда в ax число меньше 10-ти, то при его делении выдаётся ошибка и программа закрывается. Так и должно быть?

[\[Ответить\]](#)

[xrnd](#)

02-09-2010 20:57

Извиняюсь, что долго не отвечал.

Сейчас буду разбираться. Ошибка в твоей программе или в моей?

[\[Ответить\]](#)

[xrnd](#)

02-09-2010 22:57

Хорошая программа, но уж слишком запутанная 😊 Процедуры в перемешку с циклами. Ошибка в том, что DIV делит DX:AX на BX, а у тебя DX не обнуляется перед делением. Поэтому остаток от предыдущего деления попадает в старшую часть делимого. Аварийное завершение происходит потому что частное не влезает в 16 бит при таком делении.

Попробуй 3-х значные числа выводить, тоже будет косяк. Ты схитрил, взяв 1-2 значные числа и сделав дополнительную проверку)))

В моём примере делится так:

<code>xor dx,dx</code>	<i>;Обнуление старшей части двойного слова</i>
<code>div bx</code>	<i>;Деление AX=(DX:AX)/BX, остаток в DX</i>

[\[Ответить\]](#)

Кирилл

18-01-2013 23:19

Доброго времени суток.

В функции преобразования беззнакового целого в строку не вкурил следующего момента:

```
xor dx,dx ;Обнуление старшей части двойного слова
div bx ;Деление AX=(DX:AX)/BX, остаток в DX
```

.....

Остаток — в DX, а частное по идее в AX, а на следующем цикле уже получается делимое потерялось и делить будем частное от предыдущего цикла...

На TASMе 2й день пишу, так что не ругайтесь если туплю)

[\[Ответить\]](#)

Кирилл

19-01-2013 01:05

=)))) Дошло...

Извиняюсь, давно на асме(до этого контроллеры немного программировал) не писал, крыша поехала))

[\[Ответить\]](#)

fufel

04-09-2010 17:19

Да, вот это я тупанул. Надо было сразу 3-ёх значные числа задать. Поправил.

```
use16
org 100h
jmp start
;_____
n db 5
m db 3
table:
dw 355,-436,755,986,-754
dw 652,-566,768,233,966
dw -323,483,-664,534,216
vozvr_kar db 13,10,»$»
buffer rb 5
;_____
start:
mov bx,10
movzx cx,[m]
xor si,si
lp_1:
push cx
movzx cx,[n]
lp_2:
mov ax,[table+si]
call des_vid
add si,2
cmp cx,1
je end_str
call zpt
loop lp_2

end_str:
call vozvr_kar
pop cx
loop lp_1

mov ah,08h
int 21h
mov ax,4c00h
int 21h

des_vid:
push cx
push dx
xor di,di
xor cx,cx
test ax,ax
jns net_mnsa
call minus
neg ax
net_mnsa:
cmp ax,0
je decr
```

```
xor dx,dx  
div bx
```

```
add dl,'0'  
mov byte[buffer+di],dl  
inc di  
inc cx
```

```
jmp net_mnsa  
decr:  
dec di  
vivod_smv1:  
mov ah,02h  
movzx dx,[buffer+di]  
int 21h  
dec di  
loop vivod_smv1  
pop dx  
pop cx  
ret
```

```
zpt:  
push ax  
push dx  
mov ah,02h  
mov dl,', '  
int 21h  
pop dx  
pop ax  
ret
```

```
minus:  
push dx  
push ax  
mov dl,'- '  
mov ah,02h  
int 21h  
pop ax  
pop dx  
ret
```

```
vozvr_kar:  
push dx  
push ax  
mov ah,09h  
mov dx,vozwr_kar  
int 21h  
pop ax  
pop dx  
ret
```

[\[ОТВЕТИТЬ\]](#)

[xrnd](#)

06-09-2010 23:42

Эта программа лучше 😊 Но она сглючит, если одно из чисел равно 0. Попробуй сам, если не веришь.

Проблема в том, что проверка ах на 0 должна быть после деления и вывода символа.

[\[Ответить\]](#)

fufel

07-09-2010 19:23

Да, точно))) исправил. Спасибо.

[\[Ответить\]](#)

argir

23-12-2010 15:47

Я весь массив преобразовал в одну строку, поэтому процедуры всего две — ваша для преобразования(лучше не придумал) и вторая для добавления символа в строку, хотя можно было выделить еще проверку на знак.

```
use16 ;Генерировать 16-битный код
org 100h ;Программа начинается с адреса 100h
jmp start
```

```
array1 dw 2300,-7070,234,890,0,105,-9999,467,9876,-15000,876,31000
arl1 db 12
array2 rb 48;размер вычисляется arl1*4 с запасом(все числа отрицательные и пятизначные +
запятые)
start:
xor si,si
mov di,array2
mov ax,array1
movsx cx,[arl1]
pere: mov ax,[array1+si];в ax si-й элемент массива
test ax,ax;проверка на знак
jns pol;если отрицательный
mov dl,2dh ;добавляем «-»
neg ax;и инвертируем
call znak
pol: call pr2_10
cmp cx,1
jz pro ;если последняя цифра «,» не ставим
mov dl,2ch
call znak
inc si
inc si
pro: loop pere
mov dl,24h; конец строки
call znak
mov dx,array2
```

mov ah,9 ;Функция DOS 09h — вывод строки
int 21h ;Обращение к функции DOS

mov ah,8 ;Ввод символа без эха
int 21h

mov ax,4C00h ;\
int 21h ;/ Завершение программы

znak:
mov [di],dl ;Добавление элемента в строку
inc di ;Инкремент DI
ret
pr2_10:
push ax
push cx
push dx
push bx
xor cx,cx ;Обнуление CX
mov bx,10 ;В BX делитель (10 для десятичной системы)

wtuds_lp1: ;Цикл получения остатков от деления
xor dx,dx ;Обнуление старшей части двойного слова
div bx ;Деление AX=(DX:AX)/BX, остаток в DX
add dl,'0' ;Преобразование остатка в код символа
push dx ;Сохранение в стеке
inc cx ;Увеличение счетчика символов
test ax,ax ;Проверка AX
jnz wtuds_lp1 ;Переход к началу цикла, если частное не 0.

wtuds_lp2: ;Цикл извлечения символов из стека
pop dx ;Восстановление символа из стека
call znak
loop wtuds_lp2 ;Команда цикла
pop bx
pop dx
pop cx
pop ax

ret

[\[Ответить\]](#)

[xrnd](#)

24-12-2010 19:43

Написано всё правильно и работает хорошо.

В самом начале такой код:

```
mov ax,array1  
movsx cx,[ar11]
```

Первая команда, как я понял, лишняя.

Вместо второй логично использовать MOVZX или длина массива может быть отрицательной? 😊

Ну и размер array2 подходит только для твоих чисел.

Число может быть максимум 5-значным + знак + запятая или символ конца строки.

Получается `arr1 * 7`, тогда это будет «с запасом».

[\[Ответить\]](#)

argir

24-12-2010 21:20

Первые замечания в точку, а с размерами массивов не согласен, так как в первом 12 слов, а во втором 48 байт, поэтому $7/2=3,5$.)

[\[Ответить\]](#)

[xrnd](#)

24-12-2010 21:36

Тогда должна быть директива `qw`, а не `qb`.

Попробуй вывести такой массив:

```
array1 dw 12 dup(8000h)
```

[\[Ответить\]](#)

argir

26-12-2010 12:27

ASCII-символы занимают байт, зачем использовать директиву `qw`?

В задании — «массив слов со знаком», поэтому числа меньше 8000h.

[\[Ответить\]](#)

[xrnd](#)

26-12-2010 20:03

Почему меньше? `8000h = -32768`. Слово со знаком. 5 цифр и знак минус и ещё запятая. Если таких чисел 12, то получится $7*12 = 84$ ASCII-символа.

В твоей программе в этом случае «хвост» строки затрёт часть кода и на экран выведется неправильно.

[\[Ответить\]](#)

Гость

29-01-2011 17:27

```
use16
```

```
org 100h
```

```
mov bx,buffer
```

```
mov cx,[Razmer]
```

```

xor si,si
L:;-----Цыкл для каждого элемента массива
push cx
push si
mov dx,[masiv+si]
call Prozidyra
call print
pop si
pop cx
add si ,2
loop L

mov ah,08h
int 21h
mov ax,4C00h
int 21h
;-----Данные-----
bufer rb 5
znac rw 1
masiv dw 19,-210,25,10,-156
Razmer dw 5
;-----Описание процедуры -----
Prozidyra:
;----- очищаем используемые регистры-----
xor cx,cx
xor di, di
xor si, si
xor ax, ax

;-----Определяем знак сохраняем в стек
mov [znac], '+'
test dh, 10000000b
jz metca1
neg dl
xor dh,dh
mov [znac], '-'
metca1:
mov cl,dl
push [znac]
inc di
;-----Вычисляем десятичное число сохраняем в стек-----
zicol:
mov ax ,dx
sub dx ,10
jc metca
inc si
loop zicol
metca:
add ax, 30h
push ax
inc di
mov dx,si
cmp si, 10

```



```

jc metca2
xor si,si
jmp zicol
metca2:
add si, 30h
push si
inc di
push '$'
xor ax,ax

```

;Извлекаем из стёка в буфер , переворачивая цифры

```

pop ax
mov byte[bx+di],al
dec di
mov cx ,di
xor si,si
;_____
c:
pop ax
mov byte[bx+si+1],al
inc si
loop c
;_____
pop ax
mov byte[bx],al
ret
;_____
print:
xor ax ,ax
mov ah,09h
xor dx,dx
mov dx,buffer
int 21h
ret

```

[\[Ответить\]](#)

IgorKing
30-01-2011 07:35

До конца не дочитал, но...

Переменная Znac у тебя слово а должна быть байтом (ведь ты туда помещаешь символ знака, а его код — байт). И ещё:

```

...
mov [znac], '+'
test dh, 10000000b
jz metca1
neg dl ; \ по заданию же массив слов,
xor dh,dh ; / значит просто neg dx
mov [znac], '-'

```

...
А вообще, мне кажется, надо лучше откоменировать (например зачем mov cl,dl и т.д.)

[\[Ответить\]](#)

Гость
30-01-2011 18:44

переменная Znac слова , чтобы схоронить её в сёк , так это удобней.
В сёк можно сохранять только слово.

Массив слов со знаком , что значит что любое значение поместится в 8 битном регистре.
Но если использовать только 8бит нельзя будет понять положительное число или отрицательное, поэтому положительное число например 5 выглядит в dx =00 05
А если отрицательное -5 dx =FF FB
Поэтому я пришёл к выводу используется только младая часть регистра dl
dh содержит числовой бит.

neg dl ; \ по заданию же массив слов,
xor dh,dh ; / значит просто neg dx

Можно просто
neg dx ; мне проста понятней первый вариант ,)

например зачем mov cl,dl , тут тоже можно mov cx ,dx
Но если знать ,что находится в старшей части ,что в младшей части , то асоба разницы нет.
Мне так удобней.

Комментировать , не стал , пока писал понял крайне не удобно переводить числа в 10 систему исчисления вылетанием.
Но пока я её не написал я этого не знал ,)

[\[Ответить\]](#)

IgorKing
31-01-2011 15:13

Чаще мне надо ассемблер вспоминать, тогда бы догадался про Znac.

Слово это 16 бит, т.е. в 8 битном регистре оно поместится не полностью, точнее, вообще не поместится.

«...переводить числа в 10 систему исчисления вычитанием.» — а я понять не мог, что за нестандартное решение.

[\[Ответить\]](#)

[xrnd](#)
08-02-2011 17:55

Привет!

> Поэтому я пришёл к выводу используется только младая часть регистра dl
> dh содержит числовой бит.

Схитрил ты 😊 Выбрал маленькие числа для примера. Числовой бит будет только в старшем бите DH.

[\[Ответить\]](#)

Гость
02-02-2011 19:33

Решил переделать ,)
se16
org 100h

```
mov [bxz],10  
mov [cxz],8  
mov [axz],09h  
mov [dxa],Ds8
```

```
xor si,si  
xor dx,dx  
mov cx,5
```

```
cl2:  
mov dx ,word [masiv+si]  
add si ,2  
push si  
push cx  
mov [divz], dx  
call xor_xor  
call wword_to_udec_str  
call xor_xor  
call print_str  
pop cx  
pop si  
loop cl2
```

```
mov ax,4C00h  
int 21h  
masiv dw 19,-220,25,10,-156  
bxz rw 1 ; — делитель  
axz rw 1 ; — Значения ax  
dxa rw 1 ; — адрес выводимой строки  
cxz rw 1 ; - счётчик  
divz rw 1 ; - делимое  
Ds8 rb 10 ; буфер 10байт
```

процедуры

```
xor_xor: ; очищает все регистры  
xor ax,ax  
xor bx,bx  
xor dx,dx  
xor cx,cx  
xor si,si  
xor di,di  
ret
```

;Процедура вывода строки на консоль

;axz — значение регистра ax

;dxa — содержит адрес выводимой строки

print_str:

```

mov ah,byte[axz]
mov dx,[dxa]
int 21h
mov ah,08h
int 21h
ret

```

;Процедура преобразования слова в 16системи в строку в десятичном виде (без знака)

```

; bxz — делитель (10 для десятичной системы)
; divz — переменная содержит , данные для деления максимум 8байт
; Ds8 — буфер для строки (10 символов).
; cxz — счётчик

```

```

wword_to_uddec_str:

```

```

mov bx,[bxz]
mov ax,word[divz]
test ah,10000000b ; проверяет знак числа
jnz m2
jmp m3 ; если положительное переходит к метки 3
m2:
neg ax ; меняем знак если число отрицательное
mov byte[Ds8+di],'-'; добавляем минус
inc di ; смещаемся на 1 байт к концу
m3:
mov cx,[cxz] ;счётчик

```

```

zl:
div bx ;делим divz на 10
push dx ;остаток сохраняем в сёк
xor dx,dx ;очищаем так как в делении DX:AX
inc si ; si счётчик для извлечения из стёк
cmp ax,10 ; если ax меньше 10 перейти по метки
JB m1
;
loop zl ;
m1: ; если ax меньше 10
push ax ; сохраняем в стёк отдельно от цикла
mov cx,si ; цикл извлечения из стёк
pop ax ; извлекаем ax из стёка отдельно от цикла
add ax,30h ; преобразуем в строку '+30h'
mov byte[Ds8+di],al ; сохраняем в память
inc di
; смещения от начала данных
zl1: ;
pop dx ; извлекаем dx из стёка
add dx,30h ; преобразуем в строку '+30h'
mov byte[Ds8+di],dl ; сохраняем в память
inc di ; смещения от начала данных
loop zl1
;
mov [Ds8+di],',' ; добавляем запетую в конец
inc di ; смещаемся на 1 байт к концу
mov [Ds8+di],'$' ; добавляем конец строки '$'
ret

```

[\[Ответить\]](#)

[xrnd](#)

08-02-2011 18:20

Ты хорошо поупражнялся!

Без поллитра не разобраться 😊

Но я заметил некоторые недочёты в твоей программе.

Зачем тебе хог_хог? В нормальной программе это только ухудшит производительность. Вообще ты несколько раз обнуляешь регистры там, где это не обязательно. Можно просто записывать новое значение вместо старого.

Зачем «счетчик» хранится в переменной sxz? Лучше проверять частное на равенство нулю. У тебя счетчик равен 8, значит может быть 8-значное десятичное число. На самом деле слово со знаком — это максимум 5-значное десятичное число.

Короче, программа работает правильно, но её можно довольно сильно упростить и сократить 😊

[\[Ответить\]](#)

Гость

08-02-2011 22:42

Привет.

Да по упражнялся хорошо ,)

В следующей рас надо будит попробовать упростить и оптимизировать.

P.S

Пришлось немного схитрить.

Не мог понять чем (-1)от (FFFF) , отличаются , оказалось нечем .

А я думал отличия есть ,) но в регистре их нет.

Как я понял.

То есть всё что больше 8000h ,будит отрицательным число даже если оно в масиве не отрицательное а объявлена как положительное.

[\[Ответить\]](#)

[xrnd](#)

09-02-2011 13:13

В том то и дело, что объявляются просто байты или слова.

Можно интерпретировать отрицательное число со знаком как большое положительное без знака или наоборот.

Нужно внимательно выбирать команды умножения, деления и условных переходов 😊

[\[Ответить\]](#)

IgorKing

03-02-2011 14:51

Красиво...

Там где знак добавляешь вместо jnz m2 jmp m3 лучше jz m3.

[\[Ответить\]](#)

Олег

16-10-2011 11:21

Здравствуйтесь, подскажите пожалуйста, а как организовать вывод двойного слова?

[\[Ответить\]](#)

[xrnd](#)

27-10-2011 00:45

Если в десятичном виде, то такой алгоритм вызовет проблемы при делении на 10. Потому что частное может быть больше 65535, иначе переполнение.

Делить надо немного хитрее, по частям 😊

Просто будет на 32-битном ассемблере.

Нужен пример программы? Я напишу, если надо.

[\[Ответить\]](#)

Олег

07-11-2011 16:21

не отказался бы => неплохо было бы и в примере с вводом. В заранее спасибо)

[\[Ответить\]](#)

алекс

26-03-2012 21:12

```
use16
org 100h
jmp start
array dw 253,-76,94,-250,0,77,-115
length db 7
press db 13,10,'Press any key...$'
string rb 40
start:
movzx bx,[length]
xor si,si
xor di,di
mov cx,bx
string_loop:
cmp cx,0
je next
cmp cx,bx
je next
mov [string+di],','
inc di
mov [string+di],'
```

```

inc di
next:
mov ax,[array+si]
add si,2
test ax,8000h
je get_simb
mov [string+di],'- '
inc di
neg ax
get_simb:
call digit
loop string_loop
mov [string+di], '$'
mov ah,09h
mov dx,string
int 21h
mov dx,press
int 21h
mov ah,08h
int 21h
mov ax,4c00h
int 21h
;-----
digit:
push cx
push bx
push ax
mov bx,10
xor cx,cx
loop1:
xor dx,dx
div bx
push dx
inc cx
cmp ax,0
jne loop1
loop2:
pop ax
add ax,'0'
mov [string+di],al
inc di
loop loop2
pop ax
pop bx
pop cx
ret

```

[\[Ответить\]](#)

алекс
27-03-2012 13:56

;Вариант второй....лучше....

```

use16
org 100h
jmp start
array dw 253,-76,94,-250,0,77,-115
length db 7
press db 13,10,'Press any key...$'
string rb 9
start:
movzx bx,[length]
mov cx,bx
xor si,si
loop_string:
xor di,di
mov ax,[array+si]
add si,2
cmp cx,0
je next
cmp cx,bx
je next
call get_break
next:
test ax,8000h
je fore
call get_sign
fore:
call get_digit
mov ah,09h
mov dx,string
int 21h
loop loop_string
mov ah,09h
mov dx,press
int 21h
mov ah,08h
int 21h
mov ax,4c00h
int 21h
;
get_break:
mov [string+di],','
inc di
mov [string+di],','
inc di
ret
;
get_sign:
mov [string+di],'- '
inc di
neg ax
ret
;
get_digit:
push ax

```



```
push bx
push cx
xor cx,cx
mov bx,10
loop1:
xor dx,dx
div bx
push dx
inc cx
cmp ax,0
jne loop1
loop2:
pop ax
add ax,'0'
mov [string+di],al
inc di
loop loop2
mov [string+di],'$'
pop cx
pop bx
pop ax
ret
```

[\[Ответить\]](#)

nuts
06-11-2012 20:59

а у меня что-то не выводит преобразованное число. это скорей всего мой косяк, ума не хватает исправить(

[\[Ответить\]](#)

Stique
18-03-2013 14:10

А как будет выглядеть вывод двойного слова в 16-тиричном виде?

[\[Ответить\]](#)

Firefly
28-08-2013 11:32

```
use16
org 100h
jmp start
```

```
array dw 4433, 12341, -1122, -6645, 12111, 32111, -23112
measure dw 7
```

```
start:
xor di, di
mov bx, 10
mov cx, [measure]
```

```
loop_elements:
mov ax, [array+di]
add di, 2
call output_element
loop loop_elements
jmp exit
```

```
output_element:
push cx
xor cx, cx
test ax, ax
js negative_int
jns get_digit
negative_int:
push ax
xor ax, ax
mov dl, '-'
mov ah, 02h
int 21h
xor dx, dx
pop ax
neg ax
get_digit:
div bx
add dl, '0'
inc cx
push dx
xor dx, dx
cmp ax, 0
ja get_digit
mov ah, 02h
output_digits:
pop dx
int 21h
loop output_digits
mov dl, ','
int 21h
mov dl, ' '
int 21h
pop cx
xor dx, dx
ret
```

```
exit:
```

```
mov ah, 08h
int 21h
```

```
mov ax, 4C00h
int 21h
```

Где-то есть косяк, при выполнении через F9 выдается сообщение divide overflow. Однако если прогонять через отладчик, то всё проходит без ошибок, числа все выводятся. xtrnd, подскажи пожалуйста

[\[Ответить\]](#)

Firefly

28-08-2013 16:27

какие тэги использовать для оформления кода?

[\[Ответить\]](#)

Leoscoder

05-02-2014 07:03

```
use16
org 100h
jmp start
arrword dw 0ff34h,123,875,09,33,123,12ffh,-2345
start:
;Подсчет количества слов в массиве
mov ax,arrword
mov cx,start
sub cx,ax ;получаем bytes
shr cx,1 ;получаем words
xor si,si
mov bx,arrword
arrPrt:
mov ax,[bx+si]
add si,2
call sigDec
mov al,', '
call chrPrt
loop arrPrt
jmp exit
;Печатаем десятичные числа
;Печать со знаком
sigDec:
push cx
push bx
test ah,10000000b;Проверяем знак
jz unDec ;если положительный -> unDec
neg ax ;инвертируем число
push ax ;сохраняем в стеке
mov al,'-' ;печатаем «-»
call chrPrt
pop ax
;Печать без знака
unDec:
xor cx,cx
mov bx,10
lpdec:
inc cx
xor dx,dx
div bx ;в DX значение для вывода
push dx ;запоминаем в стеке.
cmp ax,0 ;проверяем на последнее вычисление
```

```

jg lpdec
decPrt:
pop ax ;берем из стека значение для вывода
add ax,48 ;корректируем с ASCII
call chrPrt
loop decPrt
pop bx
pop cx
ret
;Печатаем символ
chrPrt:
push ax
mov dl,al
mov ah,02h
int 21h
pop ax
ret
;Печатаем строку
strPrt:
push ax
mov dx,ax
mov ah,09h
int 21h
pop ax
ret
;Выход из программы
exit:
mov ax,press
call strPrt
mov ah,08h
int 21h
mov ax,4c00h
int 21h
press db 13,10,'Press any key...$'

```

[\[Ответить\]](#)

Михаил
11-12-2014 17:18

Здравствуйте! Не можете мне модернизировать прогу)) Нужно вывести байт в десятичном виде(байт должен вводиться с клавиатуры). Я сделал, чтобы сразу забитый байт выводил, а с клави не могу понять, как доделать))

code segment ; 1байт это 8 бит а числовой диапазон от 0 до 255

start:

assume cs:code

mov al,00011010b ;переносим какойто байт в al

aam ;преобразовываем из двухзначного вида в его неупакованный BCD-эквивалент

mov cx,ax

;mov ch,ah

;mov cl,al на вывод cl

```

mov al,ch
aam ;преобразовываем из двухзначного вида в его неупакованный BCD-эквиволент
mov bx,ax
;mov bh,ah на вывод bh
;mov bl,al на вывод bl

add cx,3030h ;приводим к коду в таблице ASCII
;mov ch,ah
;add bh,30h номер позиции
;mov cl,al
;add bh,30h номер позиции

add bx,3030h ;приводим к коду в таблице ASCII
;mov bh,ah
;add bh,30h номер позиции
;mov bl,al
;add bh,30h номер позиции

mov ah,02h
mov dl,bh
int 21h

mov ah, 02h
mov dl,bl
int 21h

mov ah,02h
mov dl,cl
int 21h

mov ax, 4c00h
int 21h
code ends
end start

```

[\[Ответить\]](#)

Иван
21-11-2015 23:09

Спасибо огромное!!!!!!!!!!!!!!!!!!!!!!

[\[Ответить\]](#)

Олег
25-06-2019 23:02

Здравствуйтесь!

Читая вашу статью, возникла мысль о том, как можно выводить в консоль огромные числа. Впрочем и раньше такой вопрос возникал. И хотя вы оставили ссылку, где можно узнать ответ на этот вопрос, я всё же решил сам подумать. Вспоминал деление столбиком, экспериментировал, и пришла в голову кое-какая идея. Вот решил попробовать написать прогу для вывода в консоль 128-битного числа. И какого было моё удивление, когда запустил, и прога успешно выполнялась, не зациклилась, да ещё выдала результат. Вот возникла

проблема его проверить. Виндоус калькулятор работает максимум с 8-байтными числами, а куча онлайн-калькуляторов выводит результат в экспоненциальной форме с отсечением больше половины цифр. Но первые несколько совпали. Уже хорошо. Впрочем, нашёл один сайт, на котором числа представлены в обычной форме, и там все цифры совпали. Круть! Так ведь можно число любой длины вывести на экран, внося незначительные изменения в код.

use16

org 100h

jmp start

;var

chislo db 0f7h,4ch,0fah,7bh,19h,0a6h,08h,45h,2fh,0c9h,0ddh,61h,12h,0aah,0eah,8eh

stroka rb 60

cr_lf db 13,10,'\$'

start:

xor cx,cx

xor dx,dx

mov bl,10

mov bp,chislo

mov di,15

loop2:

cmp byte[bp+di],0

jz dec_di

continue1:

mov si,di

xor ax,ax

loop1:

mov al,[bp+si]

div bl

mov [bp+si],al

dec si

jns loop1

mov dl,ah

push dx

inc cx

jmp loop2

dec_di:

dec di

js metka1

jmp continue1

metka1:

mov si,stroka

loop3:

pop dx

add dl,48

mov [si],dl

inc si

loop loop3

mov byte[si],'\$'

mov ah,9

mov dx,cr_lf

```
int 21h
```

```
mov dx,stroka
```

```
int 21h
```

```
mov dx,cr_lf
```

```
int 21h
```

```
mov ah,8
```

```
int 21h
```

```
mov ax,4c00h
```

```
int 21h
```

[\[Ответить\]](#)

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

☐ Уведомить меня о новых комментариях по email.

☐ Уведомлять меня о новых записях почтой.