Учебный курс. Часть 15. Логические операции

Автор: xrnd | Рубрика: Учебный курс | 23-04-2010 |
Распечатать запись

Логические операции выполняются поразрядно, то есть отдельно для каждого бита операндов. В результате выполнения изменяются флаги. В программах эти операции часто используются для сброса, установки или инверсии отдельных битов двоичных чисел.

Логическое И

Если оба бита равны 1, то результат равен 1, иначе результат равен 0.

AND	0	1
0	0	0
1	0	1

Для выполнения операции логического И предназначена команда AND. У этой команды 2 операнда, результат помещается на место первого операнда. Часто эта команда используется для обнуления определённых битов числа. При этом второй операнд называют *маской*. Обнуляются те биты операнда, которые в маске равны 0, значения остальных битов сохраняются. Примеры:

```
    and ax,bx           ;AX = AX & BX
    and cl,11111110b    ;Обнуление младшего бита CL
    and dl,00001111b    ;Обнуление старшей тетрады DL
```

Ещё одно использование этой команды — быстрое вычисление остатка от деления на степень 2. Например, так можно вычислить остаток от деления на 8:

```
    and ax,111b         ;AX = остаток от деления AX на 8
```

Логическое ИЛИ

Если хотя бы один из битов равен 1, то результат равен 1, иначе результат равен 0.

OR	0	1
0	0	1
1	1	1

Логическое ИЛИ вычисляется с помощью команды OR. У этой команды тоже 2 операнда, и результат помещается на место первого. Часто это команда используется для установки в 1 определённых битов числа. Если бит маски равен 1, то бит результата будет равен 1, остальные биты сохранят свои значения. Примеры:

```
    or al,dl            ;AL = AL | DL
    or bl,10000000b     ;Установить знаковый бит BL
    or cl,00100101b     ;Включить биты 0,2,5 CL
```

Логическое НЕ (инверсия)

Каждый бит операнда меняет своё значение на противоположное (0 → 1, 1 → 0). Операция выполняется с помощью команды NOT. У этой команды только один операнд. Результат помещается на место операнда. Эта команда не изменяет значения флагов. Пример:

```
    not byte[bx]        ;Инверсия байта по адресу в BX
```

Логическое исключающее ИЛИ (сумма по модулю два)

Если биты имеют одинаковое значение, то результат равен 0, иначе результат равен 1.

XOR	0	1
0	0	1
1	1	0

Исключающим ИЛИ эта операция называется потому, что результат равен 1, если один бит равен 1 или другой равен 1, а случай, когда оба равны 1, исключается. Ещё эта операция напоминает сложение, но в пределах одного бита, без переноса. 1+1=10, но перенос в другой разряд игнорируется и получается 0, отсюда название «сумма по модулю 2». Для выполнения этой операции предназначена команда XOR. У команды два операнда, результат помещается на место первого. Команду можно использовать для инверсии определённых битов операнда. Инвертируются

те биты, которые в маске равны 1, остальные сохраняют своё значение. Примеры:

```
    xor si,di           ;SI = SI ^ DI
    xor al,11110000b    ;Инверсия старшей тетрады AL
    xor bp,8000h        ;Инверсия знакового бита BP
```

Обозначение операции в комментарии к первой строке используется во многих языках высокого уровня (например C, C++, Java и т.д.). Часто XOR используют для обнуления регистров. Если операнды равны, то результат операции всегда равен 0. Такой способ обнуления работает быстрее и, в отличие от команды MOV, не содержит непосредственного операнда, поэтому команда получается короче (и не содержит нулевых байтов, что особенно нравится хакерам):

```
    mov bx,0            ;Эта команда занимает 3 байта
    xor bx,bx           ;А эта - всего 2
```

Пример программы

Допустим, у нас есть массив байтов. Размер массива хранится в байте без знака. Требуется в каждом байте сбросить 1-й и 5-й биты, установить 0-й и 3-й биты, инвертировать 7-й бит. А затем ещё инвертировать целиком последний байт массива.

```
 1 use16                 ;Генерировать 16-битный код
 2 org 100h              ;Программа начинается с адреса 100h
 3
 4     mov bx,array      ;BX = адрес массива
 5     movzx cx,[length] ;CX = длина массива
 6
 7     mov di,cx
 8     dec di
 9     add di,bx         ;DI = адрес последнего элемента
10 m1:
11     mov al,[bx]       ;AL = очередной элемент массива
12     and al,11011101b  ;Сбрасываем 1-й и 5-й биты
13     or  al,00001001b  ;Устанавливаем 0-й и 3-й биты
14     xor al,10000000b  ;Инвертируем 7-й бит
15     mov [bx],al       ;Сохраняем обработанный элемент
16     inc bx            ;В BX - адрес следующего элемента
17     loop m1           ;Команда цикла
18
19     not byte[di]      ;Инвертируем последний байт массива
20
21     mov ax,4C00h      ;\
```

```
22      int 21h             ;/ Завершение программы
23 ;-------------------------------------------------------
24 length  db 10
25 array   db 1,5,3,88,128,97,253,192,138,0
```

Упражнение

Объявите переменную *x* как двойное слово с каким-то значением. Инвертируйте 7-й, 15-й и 31-й бит. Обнулите младший байт переменной. Присвойте единичное значение битам 11-14 и 28-30. Результат сохраните в переменной *y* (естественно, она тоже должна быть объявлена как двойное слово). Инвертируйте значение *x*. Результаты можете выкладывать в комментариях.

Следующая часть »

Комментарии:

RoverWWWorm
12-05-2010 09:12

щерт, а как к примеру 30е биты инвертировать или так прописывать:
xor [n], 01000000 00000000 00000000 00000000

[Ответить]

xrnd
12-05-2010 14:43

Можно и так 🙂 Если добавить b в конце двоичного числа и без пробелов. Но удобнее в шестнадцатеричном виде записать 40000000h.

[Ответить]

RoverWWWorm
12-05-2010 17:07

aaa, запутался с этими младшими и старшими байтами, но вроде справился
use16
org 100h

xor byte[x],10000000b ;Инвертирую 7-й бит (старший бит младшего байта вроде)

```
xor byte[x+1],10000000b ;Инвертирую 15-й бит
xor byte[x+3],10000000b ;Инвертирую 31-й бит (старший бит старшего байта)

and byte[x],0 ;Обнуление младшего байта

or byte[x+1],01111000b ;Присваиваю единичное значение битам 11-14
or byte[x+3],01110000b ;Присваиваю единичное значение битам 28-30

mov ax,word[x]
mov word[y],ax

mov ax,word[x+2]
mov word[y+2],ax

not [y]

mov ax,4c00h
int 21h
;_____________
x dd 0F1111111h
y dd ?
```

[Ответить]

xrnd
12-05-2010 18:01

Вообще я имел в виду, что значение *x* надо только инвертировать. А результат всех манипуляций сохранить в *y*))) Ну ладно. Задание немного косячное, потому что если младший байт всё равно обнуляем, можно 7-й бит не инвертировать 😀

«and byte[x],0» то же самое, что «mov byte[x],0»

Кстати, ты инвертируешь не переменную x, а младшее слово y. Нужно 2 команды not на двойное слово.

[Ответить]

RoverWWWorm
12-05-2010 20:41

«Кстати, ты инвертируешь не переменную x, а младшее слово y…» Дааа малость перепутал 🙂

```
use16
org 100h

xor byte[x],10000000b ;Инвертирую 7-й бит (старший бит младшего байта вроде)
xor byte[x+1],10000000b ;Инвертирую 15-й бит
xor byte[x+3],10000000b ;Инвертирую 31-й бит (старший бит старшего байта)

and byte[x],0 ;Обнуление младшего байта

or byte[x+1],01111000b ;Присваиваю единичное значение битам 11-14
or byte[x+3],01110000b ;Присваиваю единичное значение битам 28-30

mov ax,word[x]
mov word[y],ax

mov ax,word[x+2]
mov word[y+2],ax

not [x] ; не совсем понятно, почему, «Нужно 2 команды not на двойное слово».

mov ax,4c00h
int 21h
;_____
x dd 0F1111111h
y dd ?
```

[Ответить]

RoverWWWorm
12-05-2010 21:05

aa ясно, значит not [x], только для младшего слова, тогда еще not word[x+2] нужен да.

[Ответить]

xrnd
13-05-2010 02:30

Я понял в чём дело. FASM эту команду компилирует как для 32-битного операнда 🙂 Там к команде добавляется специальный префикс и она работает как в режиме 32-битного процессора. Это будет работать на 286 процессоре и выше. Но если использовать только 16-битный код, то надо написать в 2 этапа:
not word[x]
not word[x+2]

Так что программу ты правильно написал. Её можно оптимизировать немного, если сначала загрузить переменную целиком в 2 регистра, проделать все манипуляции и затем сохранить результат. Команды будут короче, когда операнд в регистре и работать должно быстрее. Но это так, на заметку.

[Ответить]

RoverWWWorm
13-05-2010 09:22

Спасибо, учту

[Ответить]

Shindo
25-11-2010 12:04

```
use16
org 100h

mov ax,word[x]
mov bx,word[x+2]

xor al, 10000000b
xor ah, 10000000b
xor bh, 10000000b

and al, 0

or ah,01111000b
or bh,01110000b

not ax
not bx
```

mov word[y],ax
mov word[y+2],bx

mov ax, 4c00h
int 21h
;————————————————
x dd 256
y rd 1

правильно ?

[Ответить]

xrnd
25-11-2010 13:42

Да, всё правильно. Хороший вариант с загрузкой переменных в регистры 🙂

В твоём коде можно объединить 2 команды XOR в одну:

```
    xor al, 10000000b
    xor ah, 10000000b
```

заменить на

```
    xor ax,8080h
```

[Ответить]

Борис
23-12-2010 22:52

use16 ;gen 16-bit cod
org 100h ;begin 100h
;————————logik————————————————
mov ax, word[x]
mov bx, word[x+2] ; x=ax:bx
;————————8765432187654321————————
xor ax, 0100000001000000b ; 7, 15 =0
xor bx, 0100000000000000b ; 31
and al, 0 ; al=0

```
or ax, 0010010000000000b ; 11, 14 =1
or bx, 0010100000000000b ; 28, 30 =1

mov word[y], ax
mov word[y+2], bx

mov ax, word[x]
mov bx, word[x+2] ; x=ax:bx
not ax
not bx
mov word[x], ax
mov word[x+2], bx

;-------------------------------------------------

exit_progr:
mov ax,4C00h ;\
int 21h ;/ end
;-------------------------------------------------
x dd 0xffffffff
y dd ?
```

[Ответить]

xrnd
24-12-2010 19:55

Хорошо, но не совсем так, как в описании упражнения.
«Присвойте единичное значение битам 11-14 и 28-30».
Предполагается с 11 по 14 и с 28 по 30.

Дальше

```
    xor ax, 0100000001000000b ; 7, 15 =0
    xor bx, 0100000000000000b ; 31
```

Биты обычно нумеруются с нуля от младшего к старшему. Поэтому 31-й бит — самый старший.
Можно выполнить второй XOR только для BH, так как младшая часть маски нулевая.

Второй раз загружать x в регистры не нужно. Все действия выполняются с x, потом дополнительно инверсия.

[Ответить]

Борис
24-12-2010 21:49

Спасиб, понял косяки, буду стараться оптимизировать лучше

А кстати наборные маски можно делать как в си

(1<<5)||(итд итп)?

[Ответить]

xrnd
24-12-2010 22:08

Да, можно такое. Синтаксис FASM отличается от Си, но смысл тот же:

```
    xor ax,(1 shl 7) or (1 shl 15)
```

Я обычно сразу пишу в 16-ричном виде.

```
    xor ax,8080h
```

[Ответить]

Борис
24-12-2010 21:51

хочется как-то более простое решение, чем считать позиции единичек и ноликов

[Ответить]

plan4ik
01-04-2011 13:23

так вон xrnd показал
xor ax, 8080h

7Fh == 127d ; 0111 1111, и он еще не затронул знаковый разряд
80h == -127d ; 1000 0000, а этот уже знаковый

7 bit = 0x80, 1^7 степени
15 bit = 0x8000, 1^15 степени
31 bit = 0x80000000, 1^31 степени

bit value (hex)
————————-
1 0x1
2 0x2
3 0x4
4 0x8
5 0x10
6 0x20
7 0x40
8 0x80

а ваще я могу ошибатся так как теорию
сделал с xrnd по примеру «xor ax, 8080h»

[Ответить]

xrnd
01-04-2011 13:56

Ну только не 1^7, а 2^7=128

Биты нумеруются с нуля. 2 в степени номер бита.

bit value (hex)
————————-
0 0×1
1 0×2
2 0×4
3 0×8
4 0×10
5 0×20
6 0×40
7 0×80

[Ответить]

plan4ik
02-04-2011 08:02

оКэй, спасибо за поправку … я просто в win-калькуляторе коммандой (1 << lsh 1)
проверял 😉

[Ответить]

Гость
14-01-2011 00:13

use16
org 100h
mov eax,x; загружаем пameть в eax
mov ebx,[eax] ; загружаем ebx значения по адресу
not ebx ; инвертируем 7,15,31
mov Edx,ebx ; — для сохранение изменений
mov cl,[eax] ; cl влезает только последней байт x 23-31(байты)
and cl, 11110001b
or cl, 00001110b
mov dl ,cl ; сохраняем изменения в Edx
mov bh,[eax+2]; загружаем из памети со смешением 2
and bh, 11100001b
or bh, 00011110b
mov dh,bh ;
mov [y], Edx ; сохраняем изменения
mov ax,4C00h ;\
int 21h ;/ end
;————————————————————————————
x dd 0xfedcba98
y dd ?

[Ответить]

xrnd
15-01-2011 01:54

Хитрец, использовал 32-битные регистры 🙂

```
    not ebx ; инвертируем 7,15,31
```

Конечно, нужные биты инвертируются, но и остальные тоже 😀 Здесь надо использовать XOR.

Дальше тоже много ошибок.

```
    mov cl,[eax] ; cl влезает только последней байт x 23-31(байты)
    and cl, 11110001b
    or cl, 00001110b
```

Делает совсем не то, что нужно.

В CL загружается младший байт (биты 0-7).
Команда AND сбрасывает биты 1-3.
Команда OR устанавливает биты 1-3. Получается, AND делать не нужно.

[Ответить]

Гость
14-01-2011 13:21

Правильно я понял что данные в памяти компьютера хранятся , старшей бит в начале младшей в конце.

mov eax,[x] ; младшая часть a al
Если грузить значение переменой то , в регистрах они будут расположены младшая часть в младшем раздели регистра.

mov eax,x ; начала памяти указывает на старщей байт переменой x
А если использовать прямой адрес начала данных , в младшей части регистра будит страшая часть данных.

[Ответить]

xrnd
15-01-2011 02:07

Не совсем так.

Данные в памяти хранятся байтами. Это значит, что отдельный бит прочитать или записать нельзя, всегда выполняется операция с целым байтом.

В этом случае порядок битов в байте — это только обозначение. Обычно младший бит рисуют справа, а старший — слева. Нумеруют справа налево, начиная с нуля.

То, о чем ты пишешь — это порядок байтов в слове (или двойном слове) при хранении в памяти. В разных архитектурах бывает 2 варианта: little endian и big endian.

little endian — младший байт по младшему адресу — этот способ используется в архитектуре Intel x86.

big endian — старший байт по младшему адресу. Бывает и такое, но в других процессорах.

Двойное слово состоит из четырёх байтов. С регистрами всё понятно — младшая часть будет в AL, CL и т.д. На то она и младшая часть регистра.

А в памяти будет младший байт по младшему адресу, так как little endian:

```
    mov al,byte[x]   ;первый, младший байт, биты 0-7
    mov al,byte[x+1] ;второй байт - биты 8-15
    mov al,byte[x+2] ;третий байт - биты 16-23
    mov al,byte[x+3] ;четвёртый, старший байт - биты 24-31
```

[Ответить]

Гость
15-01-2011 13:50

Спасибо вроде теперь разобрался ,)

[Ответить]

Гость
15-01-2011 16:01

Спасибо теперь разобрался ,)
В регистре 123456789
В памете 987654321

[Ответить]

Гость
15-01-2011 16:03

ой не так
В регистре 0123456789
В памете 8967452301

[Ответить]

xrnd
15-01-2011 16:12

Если интересно, подробнее о порядке байтов можно почитать в википедии:
http://ru.wikipedia.org/wiki/Little_endian

[Ответить]

Гость
25-01-2011 21:20

Привет.
AND..0….1
0……0…..0
1……0…..1
Не как не пойму как ей пользоваться можно.

С такой я разобрался , а с этой прям не как ,)
0……1…..0
1……0…..0
1……1…..1
0……0…..0

[Ответить]

xrnd
26-01-2011 22:48

Привет.

Пользоваться также, как таблицей умножения 🙂
В первой строке и первом столбце значения операндов. На пересечении строки и столбца записан результат.

Например, 0 AND 0 = 0, 0 AND 1 = 0 и т.д.
А что за вторая таблица — я не понял.

[Ответить]

Андрей
31-01-2011 00:45

Как с помощью логических команд очистить (обнулить) содержимое регистра B. Возможно использование дополнительного регистра C, не

содержащего никакой полезной информации.

[Ответить]

xrnd
09-02-2011 14:37

Лучше всего использовать XOR, так как эта операция всегда даёт результат 0, если операнды равны.

```
    xor B,B
```

Ещё можно сделать AND с нулем.

```
    and B,0
```

[Ответить]

Knight212
09-02-2011 19:02

use16
org 100h

mov ax, word[x+2]
mov bx, word[x]

xor ax, 8000h
xor bx, 8080h

and bl, 0

or ax, 7000h
or bx, 7800h

mov word[y+2], ax
mov word[y], bx

not [x]

;mov ax, word[x+2]
;mov bx, word[x]

mov ax, 4C00h
int 21h

x dd 0FD597A3h
y dd ?
;After NOT
;ax = F02A
;bx = 685C

[Ответить]

xrnd
10-02-2011 15:14

Всё правильно.

Тут есть одна особенность: «not [x]» FASM сделает 32-битной командой, так как x — двойное слово.
Сейчас врядли можно найти 16-битный 8086 🙂
Но для него пришлось бы писать такой код:

```
    not word[x]
    not word[x+2]
```

[Ответить]

Shov
30-03-2011 03:19

use16
org 100h

mov ax,word[x]
mov bx,word[x+2]

xor al,80h
xor ah,128
xor bh,10000000b
xor al,al
or ah,01111000b
or bh,01110000b

mov word[y],ax
mov word[y+2],bx

not word[x]
not word[x+2]

mov ax,4C00h
int 21h

;_____________________
x dd 1234adf5h
y dd ?

[Ответить]

xrnd
01-04-2011 13:58

Всё правильно.

[Ответить]

Вика
21-04-2011 01:31

Помогите пожалуйста с программой:
Дан массив из 5 байт. Рассматривая его как массив из 8 пятиразрядных слов, найти “исключающее или” всех 8 слов для выражения “10101”.

[Ответить]

xrnd
21-04-2011 16:48

Сама-то хоть пыталась написать?

Ладно, помогу. Здесь нужно использовать сдвиги для выделения групп по 5 бит.
Так как массив всего из 5 байт, проще написать линейный алгоритм, без цикла.

```
1 use16
2 org 100h
3
4     mov si,array
5     mov ax,[si]          ; Первые два байта
6     add si,2
7     mov bl,al            ; Первое 5-битное слово
8
```

```
 9      shr ax,5
10      xor bl,al           ; 2-е слово
11
12      shr ax,3
13      mov ah,[si]         ; Чтение 3-го байта
14      inc si
15      shr ax,2
16      xor bl,al           ; 3-е слово
17
18      shr ax,5
19      xor bl,al           ; 4-е слово
20
21      shr ax,1
22      mov ah,[si]         ; Чтение 4-го байта
23      inc si
24      shr ax,4
25      xor bl,al           ; 5-е слово
26
27      shr ax,4
28      mov ah,[si]         ; Чтение 5-го байта
29      shr ax,1
30      xor bl,al           ; 6-е слово
31
32      shr ax,5
33      xor bl,al           ; 7-е слово
34
35      shr ax,5
36      xor bl,al           ; 8-е слово
37
38      and bl,1Fh          ; bl = результат
39
40      mov ax,4C00h        ; Выход из программы
41      int 21h
42
43  array db '10101'
```

[Ответить]

Вика
11-05-2011 20:30

не пойму,а где здесь сам массив из 5 байт?
Я вижу ток выражение 10101???
Почему массив заполняется этим выражением,а не допустим 10000b,11000b,10001b… и так 8 слов…(

[Ответить]

xrnd
12-05-2011 18:44

Это выражение и есть массив 🙂
Я объявил его как строку из 5 символов. Каждый символ — байт. То есть получается массив из 5 байтов.

Восемь 5-битных слов объявить не получится. Так как директива db объявляет байты. Если записать через запятую, получится 8 байтов со значениями 0001000b, 00011000b и т.д.

[Ответить]

Вика
12-05-2011 20:43

Напишите пожалуйста,коммент к каждой строке,если вас это не сильно затруднит,просто я не могу уловить алгоритм(((
Я могу сказать что делает каждая команда,а смысл уловить никак не могу(((
Плиз,оч нужна эта прога((

[Ответить]

xrnd
13-05-2011 14:27

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

    mov si,array        ; SI = адрес массива
    mov ax,[si]         ; Чтение первых двух байтов в AL и AH
    add si,2            ; Увеличить адрес на 2
    mov bl,al           ; BL = первое 5-битное слово (старшие 3 бита AL
                        ;  тоже копируются, но не влияют на результат)
    shr ax,5            ; Сдвиг на 5 бит вправо
    xor bl,al           ; Исключающее ИЛИ со вторым 5-битным словом
                        ; (старшие 3 бита не мешают)
    shr ax,3            ; Сдвиг на 3 бита вправо
    mov ah,[si]         ; Чтение 3-го байта массива
    inc si              ; Инкремент адреса
    shr ax,2            ; Сдвиг на 2 бита вправо
    xor bl,al           ; Исключающее ИЛИ со третьим 5-битным словом

    shr ax,5            ; Сдвиг на 5 бит вправо
    xor bl,al           ; Исключающее ИЛИ с четвертым 5-битным словом

    shr ax,1            ; Сдвиг на 1 бит вправо
    mov ah,[si]         ; Чтение 4-го байта массива
```

```
23      inc si              ; Инкремент адреса
24      shr ax,4            ; Сдвиг на 4 бита вправо
25      xor bl,al           ; Исключающее ИЛИ с пятым 5-битным словом
26
27      shr ax,4            ; Сдвиг на 4 бита вправо
28      mov ah,[si]         ; Чтение 5-го байта массива
29      shr ax,1            ; Сдвиг на 1 бит вправо
30      xor bl,al           ; Исключающее ИЛИ с шестым 5-битным словом
31
32      shr ax,5            ; Сдвиг на 5 бит вправо
33      xor bl,al           ; Исключающее ИЛИ с седьмым 5-битным словом
34
35      shr ax,5            ; Сдвиг на 5 бит вправо
36      xor bl,al           ; Исключающее ИЛИ с восьмым 5-битным словом
37
38      and bl,1Fh          ; Сбросить старшие 4 бита BL
39                          ; BL = результат
40      mov ax,4C00h        ; Выход из программы
41      int 21h
42
43  array db '10101'        ; Строка - массив из 5 байтов
```

Смысл алгоритма в следующем. Сдвигая по 5-бит вправо, можно выделять 5-битные слова из последовательности битов. Но так как мы работаем с байтами, нужно ещё и «собирать» эту последовательность из отдельных байтов.
То есть после сдвига на 8 бит нужно читать следующий байт из массива.

В первый раз читается сразу 2 байта. Сдвигать надо 16-битный регистр, чтобы младшие биты одного байта попадали в страшие биты другого.

Исключающее ИЛИ считается в регистре BL, при этом старшие 3 бита обнуляются в конце программы.

[Ответить]

Вика
17-05-2011 17:54

Спасибо огромнейшее!!!)))))

Теперь не слезу с этого сайта,пока асм не выучу))))

[Ответить]

annihilator
03-05-2011 16:19

Вот что получилось:
1)

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

mov ax, word[x]
mov bx, word[x+2]

xor ax, 10000000010000000b
xor bh, 10000000b
xor al, al
or ah, 01111000b
or bh, 01110000b

mov word[y], ax
mov word[y+2], bx

not word[x]
not word[x+2]

mov ax,4C00h ;\
int 21h ;/ Завершение программы
;————————————————————————————-
x dd 1
y dd ?
```

или
2)

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

mov ax, word[x]
mov bx, word[x+2]

xor ax, 10000000010000000b
xor bh, 10000000b
and al, 00000000b
or ah, 01111000b
or bh, 01110000b

mov word[y], ax
mov word[y+2], bx
```

```
not word[x]
not word[x+2]

mov ax,4C00h ;\
int 21h ;/ Завершение программы
;-------------------------------------------
x dd 1
y dd ?
```

Всё ли верно и какой вариант лучше? Или они равнозначны?

[Ответить]

xrnd
06-05-2011 00:34

Оба варианта правильные.

Разница между 1 и 2 очень небольшая.
Но наверно лучше всё-таки 1-й вариант. Команда «and al,0» содержит непосредственный операнд, который хранится в коде команды. То есть команда длиннее на 1 байт и этот байт должен быть прочитан из памяти.
«xor al,al» — работает только с регистрами, нет непосредственного операнда.

[Ответить]

алекс
09-03-2012 04:31

```
use16
org 100h

mov ax,word[x] ;â ax ìëàäøåå ñëîâî
mov dx,word[x+2] ;â dx ñòàðøåå ñëîâî
xor ax,1000000010000000b ;èíâåðòèðóåì 7é è 15é áèòû
xor dx,1000000000000000b ;èíâåðòèðóåì 31é áèò
and ax,1111111100000000b ;îáíóëÿåì ìëàäøèé áàéò x
or ax,0111100000000000b ;óñòàíàâëèâàåì áèòû ñ 11 ïî 14
or dx,0111000000000000b ;óñòàíàâëèâàåì áèòû ñ 28 ïî 30
```

```
mov word[y],ax
mov word[y+2],dx ;ñîõðàíÿåì ðåçóëüòàò

not [x]

mov ax,4c00h
int 21h ;çàâåðøåíèå ïðîãðàììû

x dd 612399522
y dd ? ;îáúÿâëÿåì ïåðåìåííûå
```

[Ответить]

алекс
09-03-2012 04:37

Я не совсем понял, в примере программы в цикле идет команда inc bx, то есть увеличение каждый раз bx на единицу. В dx у нас помещен адрес элемента массива.
Но сначала туда помещается адрес ПОСЛЕДНЕГО элемента массива, может там должно быть dec bx ?

[Ответить]

Sam
16-01-2015 03:18

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

mov ax, word[x]
mov dx, word[x+2]
xor ax, 8080h
xor dx, 8000h

xor al, al

xor ax, 7800h
xor dx, 7000h

mov word[y], ax
mov word[y+2], dx
```

```
mov ax,4C00h ;\
int 21h ;/ Завершение программы
;-------------------------------------------
x dd 0
y dd ?
```

[Ответить]

Rafael
17-08-2017 16:01

```
use16
org 100h

mov ax, word[x]
mov bx, word[x+2]
;Инвертируйте 7-й, 15-й и 31-й бит
xor ax, 8080h
xor bx, 8000h
;Обнулите младший байт переменной
xor al, al
;Присвойте единичное значение битам 11-14 и 28-30
or ax, 7800h
or bx, 7000h
;Результат сохраните в переменной y
mov word[y], ax
mov word[y+2], bx
;Инвертируйте значение x
not ax
not bx
mov word[x], ax
mov word[x+2], bx

mov ax, 4C00h
int 21h
;-----------------------
x dd 0ffffffffh
y dd 0h
```

[Ответить]

Тима
03-12-2017 13:08

помогите пожалуйста а то у меня не получается !!!!
байты элемента массива, а в котором 4-й бит имеет 1, логически сдвигаются влево на один бит. Определить сумму элементов массива

[Ответить]

Елизар
12-12-2017 05:39

use16 ;Генерировать 16-битный код
org 100h ;Программа начинается с адреса 100h
;Объявите переменную x как двойное слово с каким-то значением. Инвертируйте 7-й, 15-й и 31-й бит. Обнулите младший байт переменной. Присвойте единичное значение битам 11-14 и 28-30. Результат сохраните в переменной y (естественно, она тоже должна быть объявлена как двойное слово). Инвертируйте значение x.
xor word[x],8080h
xor word[x+2],8000h
and byte[x],00h
or word[x],7800h
or word[x+2],7000h
mov ax,word[x]
mov word[y],ax
mov ax,word[x+2]
mov word[y+2],ax
not word[x]
not word[x+2]

mov ax,4C00h ;\
int 21h ;/ Завершение программы
;_____
x dd 103cfeb0h
y rd 1

[Ответить]

Олег
10-03-2018 12:49

Вот мой код
use16
org 100h

```
mov ax,word[x]
mov dx,word[x+2]
mov bx,ax
mov cx,dx

xor al,10000000b
xor ah,10000000b
xor dh,10000000b
or ah,01111000b
or dh,01110000b
mov word[y],ax
mov word[y+2],dx

mov ax,bx
mov dx,cx
not ax
not dx
mov word[x],ax
mov word[x+2],dx

mov ax,4c00h
int 21h
;Переменные NOT00101110 01100101 10111010 01000000
x dd 0xd19a45bf ;11010001 10011010 01000101 10111111
y dd ? ;01110001 10011010 11111101 00111111
```

[Ответить]

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

Уведомить меня о новых комментариях по email.

Уведомлять меня о новых записях почтой.