Учебный курс. Часть 12. Преобразование типов

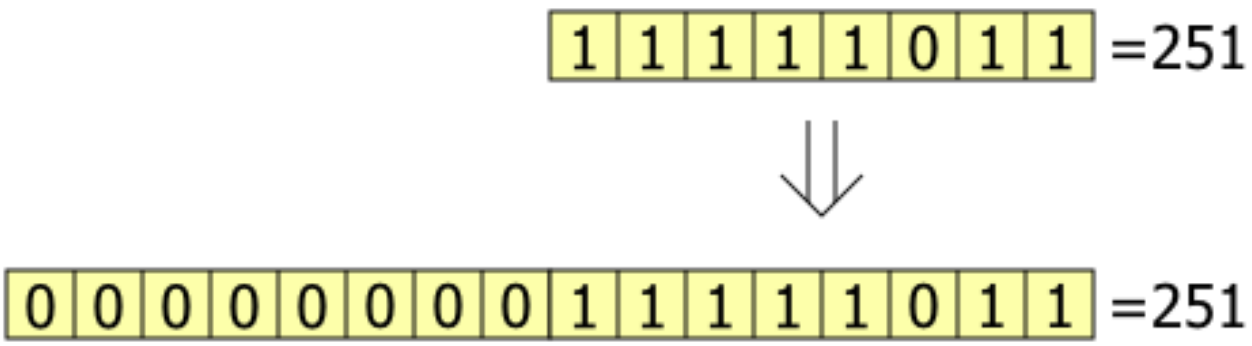
Автор: xrnd | Рубрика: Учебный курс | 18-04-2010 |
Распечатать запись

Очень часто в программах приходится выполнять действия с числами разного размера, например складывать или умножать байт и слово. Напрямую процессор не умеет выполнять такие операции, поэтому в этом случае необходимо выполнять преобразование типов. Сложность представляет преобразование меньших типов в большие (байта в слово, слова в двойное слово и т.д.)

Преобразовать больший тип в меньший гораздо проще — достаточно отбросить старшую часть. Но такое преобразование небезопасно и может привести к ошибке. Например, если значение слова без знака больше 255, то сделав из него байт, вы получите некорректное значение. Будьте внимательны, если вы используете такие трюки в своей программе.

Преобразование типов без знака

Преобразование типов выполняется по-разному для чисел со знаком и без. Для преобразования чисел без знака необходимо просто заполнить все старшие биты нулями. Например, так будет выглядеть преобразование байта в слово:



Такое преобразование можно выполнить с помощью обычной команды MOV (*x* объявлен как байт):

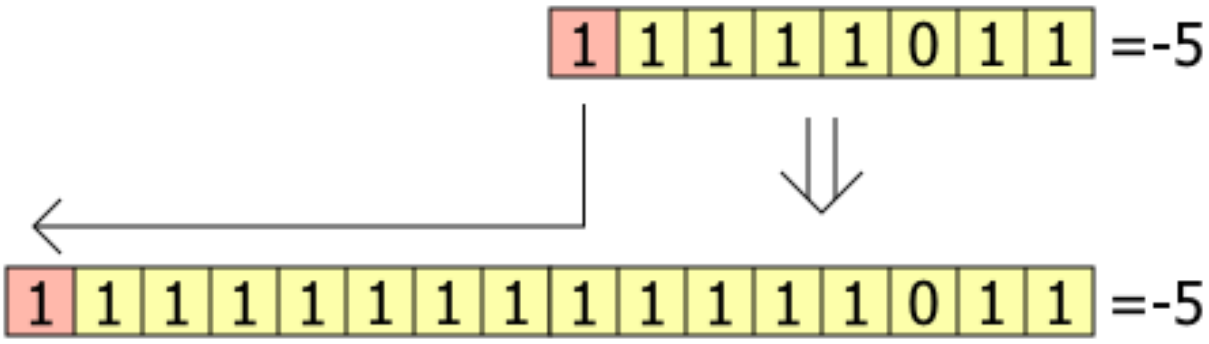
```
    mov bl,[x]
    mov bh,0      ;BX = x
;Или вот так:
    mov bl,[x]
    sub bh,bh     ;BX = x
```

Но кроме того в системе команд процессора существует специальная команда — MOVZX (копирование с нулевым расширением). Первый операнд команды имеет размер 16 бит (слово), а второй — 8 бит (байт). Тот же результат можно получить так:

```
    movzx bx,[x] ;BX = x
```

Преобразование типов со знаком

Для чисел со знаком всё немного сложнее. Если мы просто заполним старшую часть нулями, то результат будет положительным, а это не всегда верно. Поэтому преобразование выполняется путём копирования знакового бита на всю старшую часть. То есть для положительного числа со знаком старшая часть будет заполняться нулями, а для отрицательного — единицами:



Для такого преобразования предназначена команда MOVSX (копирование со знаковым расширением). Первый операнд — слово, второй операнд — байт. Например (*y* объявлен как байт):

```
    movsx cx,[y] ;CX = y
```

Существуют ещё две команды для преобразования типов со знаком: CBW (*Convert Byte to Word* — преобразовать байт в слово) и CWD (*Convert Word to Double word* — преобразовать слово в двойное слово). У этих команд

нет явных операндов. Команда CBW преобразует байт, находящийся в регистре AL, в слово в регистре AX. Команда CWD преобразует слово, находящееся в регистре AX, в двойное слово в регистрах DX:AX. Эти команды удобно использовать вместе с командами умножения и деления. Например:

```
    mov al,[y]
    cbw           ;AX = y
    cwd           ;DX:AX = y
```

Пример программы

Допустим, требуется вычислить значение формулы $x = (a + b) / c$. Все числа со знаком. Размер x — двойное слово, размер a — байт, размер b и c — слово.

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

    movsx ax,[a]        ;AX = a
    add ax,[b]          ;AX = a+b
    cwd                 ;DX:AX = a+b
    idiv [c]            ;AX = (a+b)/c, в DX остаток
    cwd                 ;DX:AX = (a+b)/c
    mov word[x],ax      ;\
    mov word[x+2],dx    ;/ x = DX:AX

    mov ax,4C00h        ;\
    int 21h             ;/ Завершение программы
;--------------------------------------------------------
a   db -55
b   dw -3145
c   dw 100
x   dd ?
```

Упражнение

Напишите программу для вычисления формулы $z = (x \cdot y) / (x + y)$. Все числа со знаком. Размер x — байт, размер y — слово, размер z — двойное слово. Проверьте работу программы в отладчике. Результаты можете выкладывать в комментариях.

Следующая часть »

Комментарии:

RoverWWWorm
20-04-2010 13:32

```
use16
org 100h

movsx ax,[x]
mov bx,ax
imul [y] ;DX:AX = AX*[y]
add bx,[y]
idiv bx ;AX =DX:AX/BX, в DX остаток
cwd
mov word[z],ax
mov word[z+2],dx

mov ax,4C00h
int 21h

;_____________
x db -23
y dw -1555
z dd ?
```

[Ответить]

xrnd
20-04-2010 20:32

Да. Всё верно.

[Ответить]

RoverWWWorm
20-04-2010 13:35

Замечательный сайт, завтра 13й урок гляну.
Спасибо за новые уроки!!!
Ураaa!!!

[Ответить]

mc-black
11-05-2010 23:19

Пример про «Например, если значение слова без знака больше 255, то сделав из него байт, вы получите некорректное значение.» на мой взгляд не очень удачный, так как если я ожидаю беззнаковый байт, то значение 255 сохранится и не потеряет смысл. если со знаком — другое дело.

[Ответить]

xrnd
12-05-2010 00:11

Написано же «если значение больше 255»)))
По сути я верно написал, но видимо не очень понятно. Я подумаю, как можно проще объяснить.

[Ответить]

mc-black
12-05-2010 20:35

А, не, все верно ты написал, это я читал невнимательно.

[Ответить]

Евгений
09-02-2012 17:23

Не подскажите как следует обрабатывать подобные моменты?

Например, я хочу произвести действия(+-*/) над знаковыми байтами и накапливать результат в EAX, а потом из EAX поместить результат куда-нибудь в память.

Если первый знаковый байт можно поместить в EAX командой movsx, то для остальных, ведь, только ADD|SUB|IMUL|IDIV будет недостаточно?

[Ответить]

Борис
19-12-2010 16:00

```
use16 ;16-bit cod
org 100h ;begin 100h
;z = (x * y) / (x + y)
mov al, [x]
cbw
imul [y] ;Dx:Ax=x*y
movsx bx, [x]
add bx,[y] ;bx=x+y
idiv bx ;ax=Dx:Ax/bx Dx ostatok
mov [z], ax
mov [zo], dx

mov ax,4C00h ;\
int 21h ;/ end
;-------------------------------------------
x db 0x32
y dw 0x32ff
z dw ?
zo dw ?
```

[Ответить]

xrnd
20-12-2010 19:29

Всё правильно, только z должно быть двойным словом (dd). Остаток, в принципе, можно не сохранять. При делении целых чисел обычно его просто игнорируют 🙂

[Ответить]

Amator
05-01-2011 00:45

может быть так:

```
use 16 ;генерируем 16 битный код
100h ; начинаем с адреса 100h

movsx ax, [x] ; AX = x
mov cx, [y] ; CX = y
imul cx ; AX = CX * AX = x * y
mov ax ; AX = x * y
```

```
cwd ; DX:AX = x * y
add cx, [x] ; CX = x + y
cwd ; CX:BX = x + y
idiv cx ; AX = x * y \ (x + y) остаток в DX
cwd ; DX :AX = x * y \ (x + y)
mov word1 [z] ax
mov word2 [z+2] dx ; z = DX :AX

mov 4c00h
int 21h
;————————————————————————
x db – 22
y dw 350
z dd ?
```

[Ответить]

xrnd
10-01-2011 23:08

Привет.

В принципе, код очень похож на правильный, но есть много мелких опечаток.
Ты хотя бы пробовал его скомпилировать? 🙂

use16 должно быть без пробела. В нескольких командах пропущены запятые между операндами. word1, word2 — непонятно, что это такое.

[Ответить]

Гость
11-01-2011 19:43

```
use16 ;Ãåíåðèðîâàòü 16-áèòíûé êîä
org 100h ;Ïðîãðàììà íà÷èíàåòñÿ ñ àäðåñà 100h

mov bh,0ffh ; bx=ff,00
mov bl,[x] ; bx=ff,[x] bx=x
mov ax,[y] ; ax =y
IMUL bx ;dx:ax=(x*y)
add bx,[y] ;dx=( x+y )
idiv bx ;(x.y)/(x+y)
```

```
mov [z],ax ;z=(x.y)/(x+y)
mov ax,4C00h
int 21h
;_____________________
x db -10
y dw -100
z dw ?
```

[Ответить]

xrnd
11-01-2011 20:42

Почти всё правильно.
Интересный способ преобразования x из байта в слово, но он подходит только для отрицательного числа. С положительным x программа будет работать неправильно.
Лучше использовать команду CBW или MOVSX.

Ещё z надо объявить с помощью директивы dd и преобразовать результат в двойное слово.

[Ответить]

Гость
12-01-2011 16:41

Понятно , вопрос возник почему для отрицательных ?
Правельноли я понял что Регистры заполняются с права налево ?
Тогда это для положительных чисел расширение можно сделать так

```
mov eax ,0 ; eax = 00000000
mov al [x] ; eax= 0000000[x]
mov [x1] ,ax;ax= 000[x]
mov [x2] ,eax;= 0000000[x]
_____________________________-
x db ; 1 байта или al
x1 dw ; раширения 1 байта до слова или ax
x2 dd ; раширения 1 байта до 2 слова или eax
```

А вот с отрицательными так не выдит , и будит неудобна раширять больше чем на 1 слова .
Хотя хз может это не правельно , может что та не так понял ,)

[Ответить]

xrnd
13-01-2011 16:54

Биты нумеруются справа налево, начиная с нуля. Поэтому младшая часть находится справа. Заполнять регистры можно как угодно, это зависит от программы.

Твой код будет работать правильно для положительных чисел, но неправильно для отрицательных. Можно, конечно, проверять знак числа и делать переход.. Но гораздо лучше использовать специальные команды CBW, CWD, MOVSX.

[Ответить]

Гость
13-01-2011 17:27

Ясна спасибо за разъяснения ,)

[Ответить]

Георгий
05-02-2011 15:38

```
use16
org 100h
; z=(x*y)/(x+y)

movsx ax,[x]
mov bx,[y]
imul bx; DX:AX=AX*BX

add bx,[x]; BX=BX+[X]

idiv bx; AX:DX=DX:AX/BX

cwd

mov word[z],ax
mov word[z+2],dx

mov 4C00h
int 21h
;—-data——
```

x db 0x20
y dw 0x20fff
z dd ?

[Ответить]

xrnd
09-02-2011 15:35

Привет.

Почти всё верно, есть небольшие опечатки. Ты пробовал компилировать эту программу? 🙂

mov 4C00h — пропущен первый операнд AX

add bx,[x] — на эту строку FASM ругается, так как размеры операндов не совпадают. Нельзя просто прибавить байт к слову.
Надо делать так:

```
    movsx cx,[x]
    add bx,cx
```

либо так:

```
    add bl,[x]
    adc bh,0
```

[Ответить]

georgmay
10-02-2011 19:17

;Я писал на смартфоне в текстовый документ, потом со смартфона на комп, затем ;сюда. Ошибок не заметил 🙂 Но я исправлюсь!
; z=(x*y)/(x+y)
use16
org 100h

movsx ax,[x]
mov bx,[y]
imul bx; DX:AX=AX*BX

```
movsx cx,[x]
add bx,cx; BX=BX+CX([X])

idiv bx; DX:AX=DX:AX/BX

cwd

mov word[z],ax
mov word[z+2],dx

mov ax,4C00h
int 21h
;—-data——
x db 0x20
y dw 0xffff
z dd ?
```

[Ответить]

xrnd
10-02-2011 20:24

Теперь все правильно, поздравляю 🙂

[Ответить]

A13R42
18-02-2011 09:20

Привет. Вот, написал, проверь пожалуйста (я так понял, остаток не надо сохранять, правильно?)

```
;———-
use16
org 100h

mov ax,[y]
movsx bx,[x]
imul bx
add bx,[y]
idiv bx
cwd
mov word[z],ax
```

mov ax,0x4c00
int 0x21

;_____________
x db 0x0F
y dw -0x0ABC
z dd ?

[Ответить]

A13R42
18-02-2011 09:23

Так. word[z] не нужен. Нужен просто [z] =)

[Ответить]

xrnd
18-02-2011 13:27

Привет.
Остаток обычно отбрасывается, сохранять не надо.

Почти всё правильно, но просто [z] работать не будет. Так как z — двойное слово, FASM выдаст сообщение, что размеры операндов не совпадают.
Для сохранения результата нужно 2 команды MOV.

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

[Ответить]

A13R42
18-02-2011 17:10

Точно! Блин, подумал, что в dx пойдёт остаток о_0. Усиленное перепрочтение предыдущей главы всё прояснило.
Большое спасибо за помощь!

[Ответить]

A13R42
18-02-2011 17:31

Хотя нет — и вправду в dx пойдёт остаток. Так, что-то я запутался окончательно. У нас частное пойдёт в ax, а остаток — в dx. При этом, в младшее слово мы сохраняем частное, а в старшее — остаток (т.е. первая половина переменной — частное, а вторая — остаток). Так зачем остаток, если ты сам сказал, что его можно отбросить? Что-то я не схватил.

[Ответить]

xrnd
18-02-2011 19:41

Остаток будет в DX после выполнения команды IDIV.
А после IDIV есть ещё команда CWD, которая преобразует слово в AX в двойное слово DX:AX. То есть в DX будет старшая часть результата.

[Ответить]

A13R42
18-02-2011 20:11

Всё, спасибо, дошло наконец-то =) Туплю что-то сегодня

[Ответить]

plan4ik
01-04-2011 10:56

```
use16
org 100h
Start:
movsx ax, byte [x]
imul [y] ; x * y

movsx si, byte [x]
add si, word [y] ; x + y

idiv si ; dx:ax / si

mov word [z], ax
mov word [z+2], dx ;z = dx:ax
Exit:
```

```
mov ax, 4c00h
int 21h

;============================

x db 0x9A
y dw 0x9BBB
z dd ?
```

[Ответить]

plan4ik
01-04-2011 11:09

Ответ в младшем слове «Z», я решил остаток тоже оставить для наглядности, малоли что … 😉

[Ответить]

xrnd
01-04-2011 16:04

Предполагалось сделать ещё преобразование слова в двойное слово Z.
А так всё правильно.

[Ответить]

plan4ik
03-04-2011 11:30

Исправил … 😉

```
use16
org 100h
Start:
movsx ax, byte [x]
imul [y] ; x * y

movsx si, byte [x]
add si, word [y] ; x + y

idiv si ; dx:ax / si
cwd
```

```
mov word [z], ax ; результат
mov word [z+2], dx ; знаковое расширение

mov ax, 4c00h
int 21h

;============================

x db 0x9A
y dw 0x9BBB
z dd ?
```

[Ответить]

xrnd
08-04-2011 18:49

Теперь совсем правильно 🙂
Кстати, byte и word писать не обязательно, если размер объявленной переменной такой же.

[Ответить]

annihilator
21-04-2011 19:20

у меня два варианта

1 примерно такой же как у всех а может и идентичен какому-то:

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

movsx ax, [x]
mov bx, ax
imul [y]
add bx, [y]
idiv bx
cwd

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

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

;_____________________________________________________
x db -55
y dw -3145
z dd ?
```

_____
_____

2 если регистры быстрее чем оперативка, то так наверное лучше:

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

movsx bx, [x]
mov cx, [y]
mov ax, bx
add bx, cx
imul cx
idiv bx
cwd

mov word[z], ax
mov word[z+2], dx

mov ax, 4C00h ;\
int 21h ;/ завершение программы
;_____________________________________________________
x db -55
y dw -3145
z dd ?
```

Во втром варианте нет «лишнего» считывания переменной(в данном случае константы) из оперативки. В дебугере проверил, кажется всё правильно.

ВОПРОСЫ:
1) как можно расчитать или узнать(с помощью какой мрограммы) скорость работы полученной программы, как узать за какое количество тактов (для одноядерного процессора) она выполняется?
2) если я правильно понял, одноядерный 32 разрядный процессор за один

такт может обработать максимум 32 бита?
3) если процессор 32 разрядный, он может за один такт выполнить две 16 битных комманды или четыре 8 битных?
4) если процессор 32 разрядный, он может за один такт выполнить оду 16 битную и половину 32 битной, а потом продолжить её выполнение в следующем такте?

[Ответить]

xrnd
24-04-2011 22:06

Оба варианта правильные.
Второй, скорее всего, будет быстрее. Но точно сказать трудно, так как зависит от многих факторов.

1) На форуме выложили такую программу на C++:
http://forum.asmworld.ru/viewtopic.php?f=2&t=130
У меня где-то валялась похожая программа на ассемблере, если найду — обязательно выложу на форум.

Чтобы сравнить 2 куска кода, можно поступить проще — выполнить кусок кода в цикле, например 1000000 раз. И засечь время.

2,3,4) Нет, все намного сложнее 🙂

32-битный процессор имеет 32-битные регистры. Поэтому большинство команд обрабатывает 32 бита.

Однако, современные процессоры содержат много хитростей, ускоряющих выполнение кода. Прежде всего, конвеер. Параллельно обрабатывается несколько команд. Например, пока выполняется сложение, процессор может вычислять адреса операндов следующей команды и читать третью команду из памяти. И всё это в одном такте! Кроме того, процессор может выполнять одновременно две команды, если они работают с разными регистрами.

16-битный код выполняется медленней на 32-битном процессоре.

Наконец, процессоры AMD и Intel устроены по-разному. Такты одного не равны тактам другого. И различаются в конкретных моделях процессоров.

Самый простой способ измерить скорость кода — это испытать его и засечь время или такты. Причём на разных процессорах 🙂 Код, оптимизированный для Intel, может выполняться медленней на AMD и наоборот.

[Ответить]

алекс
01-03-2012 04:21

```
use16
org 100h

mov bx,[y]
movsx ax,[x]
add bx,ax
imul [y]
idiv bx
cwd
mov word[z],ax
mov word[z+2],dx

mov ax,4c00h
int 21h

x db 5
y dw -20
z dd ?
```

[Ответить]

Bauka
24-03-2016 12:50

```
; z = (x*y)/(x+y)
use16
org 100h

movsx ax, [x]
imul [y]

movsx bx, [x]
add bx, [y]
```

```
idiv bx

mov word[z], ax
mov word[z+2], dx

mov ax, 4C00h
int 21h
;———————————
x db $AC
y dw -1ACh
z dd ?
```

[Ответить]

Дмитрий
06-05-2016 00:55

Мне дали вот такой пример Y=((A^3)-(C^2))/B
что то я запутался в нем, не поможете мне?
он намного сложнее чем пример

[Ответить]

Артур
05-06-2016 18:22

Тут ещё кто-нибудь обитает? Автор, статья-то в 2010 написана, а о eax,ebx и т.д. ничего не сказано. Они действительно не используются?
Мимопроходил, привет из 2016, лол

[Ответить]

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

Уведомить меня о новых комментариях по email.

Уведомлять меня о новых записях почтой.