Учебный курс. Часть 30. Команды работы с битами

Автор: xrnd | Рубрика: Учебный курс | 13-12-2010 | Распечатать запись

Работать с отдельными битами операндов можно, используя логические операции и сдвиги. Однако, кроме них в системе команд x86 существуют специальные команды для работы с битами: это команды сканирования битов и команды проверки (и модификации) битов. Впервые они появились в процессоре i386. Так что сейчас вы вряд ли найдёте процессор, в котором их нет.

Команды сканирования битов

Сканирование битов выполняется командами BSF и BSR. Эти команды очень похожи. У них 2 операнда. Первый операнд должен быть 16-битным регистром, в него записывается результат. Второй операнд может быть 16-битным регистром или словом в памяти — это обрабатываемое значение.

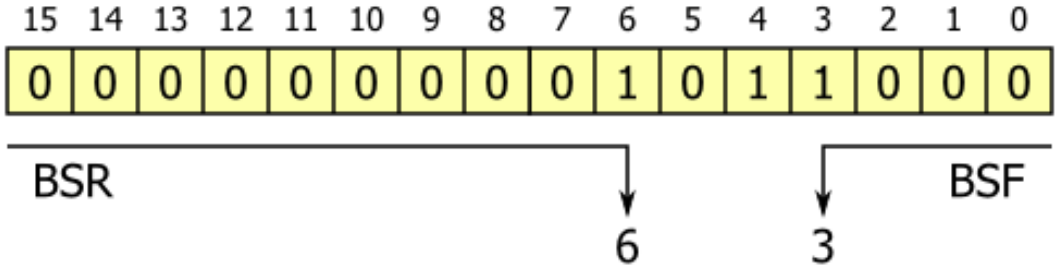
Команда BSF просматривает биты второго операнда от младшего к старшему и помещает индекс первого единичного бита в регистр. Биты нумеруются, начиная с нуля. Если единичный бит найден, то флаг нуля сбрасывается (ZF=0). Если все биты нулевые, то флаг нуля устанавливается (ZF=1), а значение первого операнда будет неопределённым (на разных процессорах может быть по-разному). Например, мой процессор (Athlon XP) в этом случае не изменяет значение в регистре. Пример кода:

```
    mov ax,01011000b  ;AX=58h
    bsf bx,ax         ;BX=3, ZF=0
    xor ax,ax         ;AX=0
    bsf bx,ax         ;BX=?, ZF=1
```

Команда BSR отличается тем, что просматривает биты от старшего к младшему. Всё остальное также.

```
    mov ax,01011000b  ;AX=58h
    bsr bx,ax         ;BX=6, ZF=0
    xor ax,ax         ;AX=0
    bsr bx,ax         ;BX=?, ZF=1
```

Картинка для наглядности:



Команды проверки и модификации битов

Команда BT копирует значение проверяемого бита в флаг CF. Вот и вся проверка! После этого можно выполнить условный переход командами JC или JNC, в зависимости от значения бита. У команды два операнда: слово в регистре или в памяти и номер бита, который может находиться в регистре или быть непосредственным значением. Примеры использования команды:

```
    bt ax,0          ;Проверка младшего бита AX
    jc m1            ;Переход, если бит равен 1
    mov cx,3         ;CX=3
    bt ax,cx         ;Проверка 3-го бита AX
    jnc m1           ;Переход, если бит равен 0
```

Ещё 3 команды немного отличаются от BT:

- команда BTR проверяет бит и затем сбрасывает его;
- команда BTS проверяет бит и затем устанавливает его в 1;
- команда BTC проверяет бит и затем инвертирует его.

Эти команды удобны тем, что можно совместить проверку бита и присвоение ему нового значения.

Пример программы

Команду BT можно использовать в цикле для вывода чисел в двоичном виде. Вместо команды условного перехода я использовал команду ADC. Она прибавляет значение бита к коду символа. Получается ‘0’ или ‘1’. В коде используются макросы из предыдущей части.

```
 1 include 'proc16.inc'
 2 use16                         ;Генерировать 16-битный код
 3 org 100h                      ;Программа начинается с адреса 100h
 4     jmp start
 5 ;--------------------------------------------------------------------------
 6 ; Данные
 7 s_1   db ' 1234h = $'
 8 s_2   db '0ABCDh = $'
 9 s_pak db 'Press any key...$'
10 ;--------------------------------------------------------------------------
11 start:
12     stdcall print,s_1          ;Вывод строки s_1
13     stdcall print_bin,1234h    ;Вывод числа 1234h
14     stdcall endline            ;Переход на новую строку
15     stdcall print,s_2          ;Вывод строки s_2
16     stdcall print_bin,0ABCDh   ;Вывод числа 0ABCDh
17     stdcall endline            ;Переход на новую строку
18     stdcall print,s_pak        ;Вывод строки 'Press any key...'
19     mov ah,8                   ;\
20     int 21h                    ;/ Ввод символа без эха
21     mov ax,4C00h               ;\
22     int 21h                    ;/ Завершение программы
23
24 ;--------------------------------------------------------------------------
25 ; Процедура вывода числа в двоичном виде
26 proc print_bin uses ax cx bx dx, value
27     mov bx,[value]             ;Загрузка числа в BX
28     mov cx,15                  ;Счетчик битов
29     mov ah,2                   ;Для функции DOS 02h
30 @@: mov dl,'0'                 ;DL='0'
31     bt bx,cx                   ;Проверка бита!
32     adc dl,ch                  ;Прибавление значения бита (CH=0)
```

```
33      int 21h                     ;Вывод символа
34      dec cx                      ;Декремент счетчика
35      jns @b                      ;Переход, если неотрицательное значение
36      ret                         ;Возврат из процедуры
37  endp
38
39  ;-------------------------------------------------------------------------
40  ; Процедура вывода строки
41  proc print uses ax dx, str
42      mov ah,9                    ;Функция DOS 09h
43      mov dx,[str]                ;Загрузка адреса строки в DX
44      int 21h                     ;Вывод строки
45      ret                         ;Возврат из процедуры
46  endp
47
48  ;-------------------------------------------------------------------------
49  ; Процедура вывода перехода на новую строку
50  proc endline
51      call @f                     ;Подумайте, как это работает
52      db 13,10,'$'
53  @@: call print
54      ret
55  endp
```

Результат работы программы:

Упражнение

Если вам ещё не надоели упражнения… Напишите процедуру, которая сбрасывает старший единичный бит и устанавливает в единицу младший нулевой бит в регистре AX (лучше прочитать ещё раз). Результаты можете писать в комментариях или на форуме.

Следующая часть »

Комментарии:

IgorKing
14-12-2010 16:54

Без макросов (не нравятся они мне).

Proc:

```
btr ax,15
bts ax,0

ret
```

Или так:

Proc:

```
and ah,01111111
or al,00000001

ret
```

Хм, какие-то они маленькие, может я задание не понял?

[Ответить]

xrnd
15-12-2010 21:33

Задание не очень понятно написано.

Я имел в виду следующее. Допустим, есть число 00011001.
Старший единичный бит — 4-й. Его надо сбросить — получится 00001001.
Младший нулевой бит — 1-й. Его надо установить — получится 00001011.

[Ответить]

IgorKing
16-12-2010 13:29

Proc:

```
bsr cx,ax
btr cx,ax

bsf cx,ax
bts cx,ax

ret
```

[Ответить]

xrnd
16-12-2010 14:35

А в отладчике смотрел, как это работает?

Не всё так просто 🙂
У BTR и BTS операнды должны быть наоборот. Номер бита — второй операнд.
Команда BSF ищет единичные биты, а нужно найти нулевой.

И ещё неплохо бы проверить флаг нуля, вдруг нет ни одного единичного бита? Тогда неправильно будет работать.

[Ответить]

IgorKing
16-12-2010 14:59

Так, сейчас всё ещё раз перечитаю и сделаю…

Proc:

```
bsr cx,ax
jz @f
btr ax,cx

@@:

mov cx,15

@@:
bt ax,cx
jc Bit_1
bts ax,cx
jmp .Exit
Bit_1:
loop @b

.Exit
ret
```

[Ответить]

xrnd
16-12-2010 15:26

Вот, это уже хороший вариант 🙂
Но только цикл начинается со старшего бита и самый младший он не проверит.
Лучше сделать его от 0 до 15 с помощью условного перехода вместо команды LOOP.

[Ответить]

IgorKing
14-12-2010 19:06

Кстати, просмотрел твой пример и завис на последней процедуре. Объясни, пожалуйста, я-то думал что db,dw и.т.д. выполняются только при компиляции, а если нет то до неё всё равно не дойдёт.

[Ответить]

xrnd
15-12-2010 21:35

Директивы данных компилируются, как и команды.
Можно прямо среди кода добавлять любые данные. У меня это 3 символа. Они не выполняются, как код, потому что перед ними стоит команда CALL.

[Ответить]

IgorKing
16-12-2010 13:38

Ну да, это я понял; так тоже можно команды писать, вместо inline.
Я вот только не понял что получиться: вызывается endline и сразу print. Куда и как попадает 13,10,'$' процедура ж вроде в одном месте в памяти находится?

[Ответить]

xrnd
16-12-2010 14:38

Команда CALL @f помещает в стек адрес возврата, который будет равен следующему адресу после самой команды 🙂

[Ответить]

IgorKing
16-12-2010 15:15

Я тодохну, а потом ещё над этим подумаю…

[Ответить]

IgorKing
16-12-2010 17:35

Дошло! Ты с помощью call @f сначала задаёшь адрес 13,10,'$' для [str], а потом печатаешь переход на новую строку… какое клёвое извращение!

[Ответить]

xrnd
17-12-2010 16:58

Это не я придумал, трюк известный. Команда эта используется для помещения адреса строки в стек 🙂

[Ответить]

Станислав
26-12-2010 20:12

Скажи, пожалуйста, вот существует директива file 'filename', а как её можно использовать, после присоединения файла программа перестаёт работать. Или она существует для присоединения ресурсов?

[Ответить]

xrnd
27-12-2010 21:33

Эта директива добавляет байты из указанного файла, как если бы они были объявлены директивой db. Скорей всего, неправильно присоединяешь, поэтому не работает.

Использовать можно, например, чтобы включить большой кусок текста в исполняемый файл. Вместо того, чтобы забивать текст в исходник на ассемблере, можно создать текстовый файл и воспользоваться директивой file.

[Ответить]

Станислав
28-12-2010 15:03

А в какую же тогда часть вставить директиву file ‘file_name’ и нужно ли ещё к ней добавлять что-то, чтобы программа не переставала работать и этот текст можно было бы использовать в программе, с db то оно проще, но для небольших фрагментов текста?

[Ответить]

xrnd
29-12-2010 22:10

Добавлять директиву file нужно в ту же часть программы, что и другие переменные. Это если использовать её для вставки текста из файла. В COM-программе переменные находятся после кода или в начале, но перед ними ставится команда JMP.

Чтобы было понятнее, приведу пример:

```
    jmp start
hello db 'Hello, world!$'
start:
    ... ; Здесь код
```

Если записать строку в файл hello.txt, то можно будет тоже самое сделать так:

```
    jmp start
hello file 'hello.txt'
start:
    ... ; Код
```

[Ответить]

argir
09-01-2011 19:24

Вот ,что у меня получилось:

; ————————-

proc work_bin uses bx dx, value
mov ax,[value]

```
bsr bx,ax; в bx номер 1-ой левой единицы
jz @f;если нет 1, то переход
neg ax; инвертируем ax
bsf dx,ax; в dx номер 1-ой правой единицы, в прошлом 0
neg ax;возвращаем ax
btc ax,bx;\
btc ax,dx;/преобразуем ax
jmp p1
@@: inc ax
p1: ret
endp
```

[Ответить]

xrnd
11-01-2011 10:18

Очень хорошо, но есть одна ошибка.
Вместо команды NEG здесь нужно использовать NOT.
Различие между этими командами в том, что NOT инвертирует все биты, а NEG инвертирует знак числа (преобразует в дополнительный код).
NEG 00000101b (5) -> 11111011b (-5)
NOT 00000101b (5) -> 11111010b (-6)

Ещё неплохо бы проверить флаг нуля после команды BSF.

[Ответить]

Гость
01-03-2011 16:43

```
use16
org 100h
mov cx ,8
mov ax ,1101b
mov si,0 ; смещение от 0 байта
@@:
bt ax ,si ; переносим в флаг c , байта со смешением si
jnc @f ; если флаг c = 0 , делаем переход
inc si ; смещаемся на 1 байт к старщем битам
loop @b
; если нулевых байт нет не чего меняться не будит
@@:
inc si
xchg cx, si ; смешение от начала , используем для сдвига
RCR ax , cl ; данное смешение использует флаг c
; после смешения в флаге c , оказывается необходимый бит
CMC ;инвертируем
RCL ax , cl ; двигаем обратно , чтобы число было корректным
; ———— сбросить старщей бит
mov dx, 16 ; это для вынесения сдвига
bsr bx , ax ; узнаём позицию старщего бита , относительно 0 бита.
sub dx,bx ; узнаём позицию старщего бита , относительно самого старшего бита
xchg cx, dx
```

RCL ax , cl ; старщей бит оказывается в флаге c
CMC ; меняем на противоположный
RCR ax , cl
m1:
mov ax,4C00h
int 21h

[Ответить]

xrnd
02-03-2011 13:20

Хорошо, но уж очень сложно.

Например

```
    RCR ax , cl ; данное смешение использует флаг c
    ; после смешения в флаге c , оказывается необходимый бит
    CMC ;инвертируем
    RCL ax , cl ; двигаем обратно , чтобы число было корректным
```

Можно заменить одной командой BTC или BTS.

Младший нулевой бит тоже можно искать проще. Например, сделать NOT ax и найти индекс младшего единичного бита командой BSF.

[Ответить]

Гость
02-03-2011 16:58

Ясна , спасибо .
BTC — куда удобней ,)
А так куда проще.
bsr bx,ax
btc ax,bx
not ax
bsf bx,ax
btc ax,bx
not ax

[Ответить]

xrnd
03-03-2011 13:28

Ещё хорошо было бы проверить флаг ZF после команд BSR и BSF.

Например, если AX=0, то твой код будет неправильно работать.

[Ответить]

RoverWWWorm
27-04-2011 18:50

```
include ‘c:\fasm\include\macro\proc16.inc’

use16
org 100h

start:

stdcall myproc,1234h
mov ax,4C00h
int 21h

;———————
proc myproc uses ax cx,n
mov ax,[n]
bsr cx,ax ;смотрим страший единичный бит, и сохраняем в cx
btr ax,cx ;сбрасываем его

xor cx,cx
@@:
bt ax,cx ;проверяем младший бит в ax
jnc @f ;если он равен 0, то переходим на метку
inc cx ;иначе переходим на следующий бит
cmp cx,16 ;если проверили все биты,
je exit ; то выходим
loop @b
@@:
bts ax,cx ;проверяем бит и затем устанавливаем его в 1
exit:
ret
endp
```

[Ответить]

xrnd
06-05-2011 00:18

Старший единичный бит сбрасывается правильно, хотя неплохо бы ещё проверить флаг ZF после команды BSR.

Цикл с ошибкой написан. Команда LOOP выполнится один раз, так как в первой итерации CX=1. Надо заменить на JMP.

[Ответить]

RoverWWWorm
16-05-2011 20:21

даа, сглупил. Ну тогда если заменить loop на jmp все нормально будет рабодать?

[Ответить]

xrnd
18-05-2011 00:50

Да, вроде правильно будет.

[Ответить]

fufel
30-05-2011 19:37

Здравствуте!

include ‘proc16.inc’
include ‘mymacro.inc’

use16
org 100h

stdcall bit,00010011b
exit

proc bit, bin
mov ax,[bin]
bsr bx,ax
btr ax,bx
mov cx,15
xor bx,bx
@@:
bt ax,bx
jnc @f
inc bx
dec cx
loop @b
@@:
bts ax,bx
ret
endp

[Ответить]

xrnd
23-06-2011 15:46

Всё правильно, но нет проверки флага ZF. Если bin=0, то будет ошибка.

[Ответить]

алекс
21-07-2012 09:14

;вот мой вариант, может корректно обрабатывать все числа и вида 000111111
;включительно. Проверено в отладчике.
include ‘PROC16.INC’

use16
org 100h
mov ax,63h
stdcall scan
mov [num],ax

```
mov ax,4c00h
int 21h
;————————————-процедура————————-
proc scan uses bx dx cx
bsr dx,ax ;сканируем биты от старшего
jnz @f ;переход если есть хоть один единичный
.incr: ;если все нулевые то
bts ax,0 ;устанавливаем нулевой бит в единицу и
jmp .fin ;переходим на выход из процедуры
@@: ;если есть хоть один единичный то
btr ax,dx ;сбрасываем старший единичный в «0»
bsf bx,ax ;сканируем биты от младшего
cmp bx,0 ;проверяем нулевой бит и
jne .incr ;переходим на метку если он равен «0», а если «1» то
xor cx,cx ;обнуляем регистр CX
@@: ;метка цикла
cmp cx,15 ;проверка счетчика цикла по макс. значению
je .fin ;выход из цикла
inc cx ;никримент четчика цикла
cmp cx,dx ;проверка счетчика для пропуска ранее сброшенного бита
je @b ;переход на метку цикла для инкримента счетчика
bt ax,cx ;проверка значения текущего бита
jc @b ;переход на метку цикла если текущий бит равен «1»
bts ax,cx ;установка первого нулевого бита в «1»
.fin: ;метка выхода из процедуры
ret
endp
;————————————————————————————-
num dw ?
```

[Ответить]

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Добавить

- [ ] Уведомить меня о новых комментариях по email.
- [ ] Уведомлять меня о новых записях почтой.