Описание системы команд микропроцессоров Intel

Материал, приведенный в данном разделе справочной системы, связан с уроком 6, на котором мы рассматривали формат машинной команды микропроцессора и систему его команд в целом.

Выберите тему:

aaa	aad	aam	aas	adc	add	and
bound	bsf	bsr	bswap	bt	btc	btr
bts	call	cbw	cwde	clc	cld	cli
cmc	cmp	cmps/cmpsb /cmpsw/cmpsd	cmpxchg	cwd	cdq	daa
das	dec	div	enter	hlt	idiv	imul
in	inc	ins/insb /insw/insd	int	into	iret/iretd	jcc
jcxz	jecxz	jmp	lahf	lds	les	lfs
lgs	lss	lea	leave	lgdt	lidt	lods/lodsb /lodsw/lodsd
loop	loope	loopz	loopne	loopnz	mov	movs/movsb /movsw/movsd
movsx	movzx	mul	neg	nop	not	or
out	outs	pop	popa	popad	popf	popfd
push	pusha	pushad	pushf	pushfd	rcl	rcr
rep/repe/repz /repne/repnz	ret/retf	rol	ror	sahf	sal	sar
sbb	scas/scasb /scasw/scasd	setcc	sgdt	sidt	shl	shld
shr	shrd	stc	std	sti	stos/stosb /stosw/stosd	sub
test	xadd	xchg	xlat/xlatb	xor	_	_

Порядок описания команд будет следующим:

- название команды с расшифровкой ее мнемонического обозначения — это облегчит процесс запоминания и последующего использования команды в соответствии с ее функциональным назначением;
- синтаксическое описание команды, поясняющее возможные сочетания операндов для данной команды. При этом сложные синтаксические описания будут приведены в виде синтаксических диаграмм, что позволит в наиболее компактной форме изобразить все возможные сочетания операндов;
- состояние флагов после выполнения команды;
- описание типового применения команды с примером и (или) ссылка на урок, в котором демонстрируется пример применения команды;
- номера занятий и приложений, а также список команд, которые функционально связаны с данной командой.

Для описания команд приняты обозначения:

1. Для описания состояния флагов после выполнения некоторой команды будем использовать выборку из таблицы, отражающей структуру регистра флагов `eflags`:

31	18	17	16	15	14	1312	11	10	09	08	07	06	05	04	03	02	01	00
0	0	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

2. В нижней строке этой таблицы приводятся значения флагов после выполнения команды. При этом используются следующие обозначения:
    - *1* — после выполнения команды флаг устанавливается (равен 1);
    - *0* — после выполнения команды флаг сбрасывается (равен 0);
    - *r* — значение флага зависит от результата работы команды;
    - *?* — после выполнения команды флаг не определен;
    - пробел — после выполнения команды флаг не изменяется;
3. Для представления операндов в синтаксических диаграммах используются следующие обозначения:
    - *r8, r16, r32* — операнд в одном из регистров размером байт, слово или двойное слово;
    - *m8, m16, m32, m48* — операнд в памяти размером байт, слово, двойное слово или 48 бит;
    - *i8, i16, i32* — непосредственный операнд размером байт, слово или двойное слово;
    - *a8, a16, a32* — относительный адрес (смещение) в сегменте кода.
4. На многих диаграммах в целях компактности возможные сочетания операндов показаны в виде следующей конструкции:

Конструируя команду на основе подобной синтаксической диаграммы, вы должны помнить о соответствии типов. В подобной диаграмме допустимы только следующие сочетания: "r8, m8", "r16, m16", "r32, m32". Например, сочетание "r8, m16" недопустимо. Однако есть единичные случаи, когда подобные сочетания возможны; тогда они специально оговариваются.

5. Описанная в данном приложении система команд в полном объеме поддерживается микропроцессором Pentium. Предыдущие модели микропроцессора могут не поддерживать отдельные команды. Чтобы прояснить этот момент, мы будем указывать в примерах для каждой команды директиву типа .286. Это будет означать, что описываемая команда поддерживается всеми моделями микропроцессора, начиная с i286. Если ничего не указывается, то это означает, что данная команда работает на всех моделях микропроцессоров Intel, начиная с i8086/8088.

AAA

(Ascii Adjust after Addition)
ASCII-коррекция после сложения

Схема команды: aaa

Назначение: корректировка неупакованного результата сложения двух одноразрядных неупакованных BCD-чисел.

Синтаксис
Алгоритм работы:

- проанализировать значение младшего полубайта регистра al и значение флага af;
- если (значение младшего полубайта регистра al >9) или (AF=1), то выполнить следующие действия:
  - увеличить значение al на 6;
  - очистить старший полубайт регистра al;
  - увеличить значение ah на 1;
  - установить флаги: af = 1, cf = 1,

  иначе сбросить флаги af = 0 и cf = 0.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	?	?	r	?	r

Применение:
Обычно команда aaa используется после сложения каждого разряда распакованных BCD-чисел командой add. Каждая цифра неупакованного BCD-числа занимает младший полубайт байта. Если результат сложения двух одноразрядных BCD-чисел больше 9, то число в младшем полубайте результата не есть BCD-число. Поэтому результат нужно корректировать командой aaa. Эта команда позволяет сформировать правильное BCD-число в младшем полубайте и запомнить единицу переноса в старший разряд путем

увеличения содержимого регистра ah на 1.
К примеру, сложить два неупакованных BCD-числа: 08 + 05:

```
        mov     ah,08h  ;ah=08h
        mov     al,05h  ;al=05h
        add     al,ah
;al=al+ah=05h+08h=0dh – не BCD-число
        xor     ah,ah   ;ah=0
        aaa             ;ah=01h,al=03h
– результат скорректирован
```

См. также: урок 8, приложение 7 и команды aad, aam, aas, daa, das

AAD

(Ascii Adjust before Division)
ASCII-коррекция перед делением

Схема команды: aad
Назначение:

- подготовка двух неупакованных BCD-чисел для операции деления;
- преобразование двузначного неупакованного BCD-числа меньшего 63h (99_{10}) в двоичное представление.

Синтаксис
Алгоритм работы:

- умножить значение регистра ah на 10 и сложить полученное значение с содержимым регистра al: (ah*10)+al;
- присвоить регистру al значение (ah*10)+al;
- обнулить регистр ah.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	r	r	r	r	?

Применение:
Команду aad используют для подготовки двузначного неупакованного BCD-числа в регистре ax для операции деления. Так как в системе команд микропроцессора нет команды деления для BCD-чисел, такое число нужно предварительно преобразовать в двоичный вид. Для этого старший разряд двузначного BCD-числа помещается в регистр ah, умножается на 10 и складывается с разрядом единиц двузначного BCD-числа 9 в регистре al. В результате этих действий и получается соответствующее двоичное число в регистре ax. Далее в программе уже можно применять обычную команду деления div, оперирующую двоичными данными. Команду aad можно применять и просто для преобразования неупакованного двузначного BCD-числа в его двоичный эквивалент. Есть еще интересный момент — если посмотреть на коды символов шестнадцатеричных цифр в таблице ASCII, то видно, что они похожи на BCD-числа. Исключение составляет лишь значение старшей тетрады (для BCD-числа это так называемая зона с нулевым значением) - оно равно 3. Можно сделать вывод, что если предварительно обнулить значение старшей

тетрады для кодов двух символов (от 0 до 9), то эту команду вполне можно применять и для преобразования двузначных десятичных чисел в символьном представлении в их двоичный эквивалент, что и отражено в названии команды. Для иллюстрации рассмотрим два примера.

Пример 1. Разделить десятичное число 18 на 9. Подготовить результат к выводу на экран.

```
        mov     ah,01h  ;ah=01h
        mov     al,08h  ;al=08h =>
ax=0108h
        mov     bl,09   ;bl=09h
        aad             ;al=12h —
двоичный эквивалент десятичного числа
18
        div     bl      ;al=02h,ah=00h
        or      al,30h  ;al=32h —
ASCII-представление числа 2, можно
выводить на экран
```

Пример 2. Преобразовать десятичное число 16 в символьном виде в эквивалентное двоичное число.

```
        mov     ax,3136h
;ax=3136h
        and     ax,0f0fh
;ax=0106h
        aad             ;al=10h —
получили его двоичный эквивалент
```

См. также: уроки 3, 8, приложение 7 и команды aaa, aam, aas, daa, das

AAM

(Ascii Adjust after Multiply)
ASCII-коррекция после умножения

Схема команды: aam

Назначение:

- корректировка результата умножения двух неупакованных BCD-чисел;
- преобразование двоичного числа меньшего 63h (99_{10}) в его неупакованный BCD-эквивалент.

Синтаксис

Алгоритм работы:

- разделить значение регистра al на 10;
- записать частное в регистр ah, остаток — в регистр al.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	r	r	r	r	?

Применение:
Команду aam используют для коррекции результата умножения двух неупакованных BCD-чисел. Специальной команды умножения BCD-чисел нет. Поэтому BCD-числа умножаются поразрядно, как обычные двоичные числа, командой mul. Максимальное число, которое получается при таком умножении, — это $9*9=81_{10}=51_{16}$. Отсюда понятно, что значения, для которых командой aam можно получить их двузначный BCD-эквивалент в регистре ax, находятся в дипазоне от 00h до 51h. Эту команду можно применять и для преобразования двоичного числа из регистра ax (в диапазоне от 0 до 63h) в его десятичный эквивалент(соответственно, из диапазона от 0 до 99_{10}).

Пример 1. Умножить десятичное число 8 на 9. Подготовить результат к выводу на экран.

```
        mov     ah,08h  ;ah=08h
        mov     al,09h  ;al= 09h
        mul     ah      ;al=48h —
двоичный эквивалент 72
        aam             ;ah=07h,al=02h
        or      ax,3030h
;ax=3732h — ASCII-представление числа
72
```

Пример 2. Преобразовать двоичное число 60h в эквивалентное десятичное число.

```
;поместим число 60h в регистр ax
        mov     ax,60h  ;ax=60h
        aam             ;ax=0906h —
получили десятичный эквивалент числа
60h
        or      ax,3030h
;символьный эквивалент, можно выводить
на экран
```

См. также: урок 8, приложение 7 и команды aaa, aad, aas, daa, das

AAS

(Ascii Adjust after Substraction)
ASCII-коррекция после вычитания

Схема команды: aas
Назначение: корректировка результата вычитания двух неупакованных одноразрядных BCD-чисел.

Синтаксис
Алгоритм работы:
если (младший полубайт регистра al меньше 9) или (флаг af=1), то выполнить следующие действия:

- уменьшить значение младшего полубайта регистра al на 6;
- обнулить значение старшего полубайта регистра al;
- установить флаги af и cf в 1;

иначе установить флаги af и cf в 1.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	?	?	r	?	r

Применение:
Команду aas используют для коррекции результата вычитания двух неупакованных одноразрядных BCD-чисел после команды sub. Операндами в команде sub должны быть правильные одноразрядные BCD-числа. Рассмотрим возможные варианты вычитания одноразрядных BCD-чисел:

- 5-9 — для вычитания необходимо сделать заем в старшем разряде. Факт такого заема в микропроцессоре фиксируется установкой флагов cf и af в 1 и вычитанием 1 из содержимого ah. В результате после команды aas в регистре al получается правильное значение (модуль результата), которое для нашего примера (с учетом заема из старшего разряда) составляет 6. Одновременно моделируется заем из старшего разряда, что позволяет производить вычитание длинных чисел.
- 8-6 — для вычитания нет необходимости делать заем в старшем разряде. Поэтому производится сброс флагов cf и af в 0, а ah не изменяется. В результате после команды aas в регистре al получается правильное значение (модуль результата), которое для нашего примера составляет 2.

Пример 1. Вычесть десятичное число 8 из 5. Подготовить результат к выводу на экран.

```
        mov     al,05h
        mov     bl,08h
        sub     al,bl   ;al=0fdh
        aas             ;al=07, cf=af=1
        or      al,30h  ;al=37h – код
символа 7
;вывод результата на экран
        mov     ah,2
        mov     dl,al
        int     21h
```

См. также: уроки 3, 8, приложение 7 и команды aaa, aad, aam, daa, das

ADC

(Addition with Carry)
Сложение с переносом

Схема команды: adc приемник,источник
Назначение: сложение двух операндов с учетом переноса из младшего разряда.

Синтаксис
Алгоритм работы:

- сложить два операнда;
- поместить результат в первый операнд: приемник=приемник+источник;
- в зависимости от результата установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	02
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команда adc используется при сложении длинных двоичных чисел. Ее можно использовать как самостоятельно, так и совместно с командой add. При совместном использовании команды adc с командой add сложение младших байтов/слов/двойных слов осуществляется командой add, а уже старшие байты/слова/двойные слова складываются командой adc, учитывающей переносы из младших разрядов в старшие. Таким образом, команда adc значительно расширяет диапазон значений складываемых чисел. В приложении 7 приведен пример программы сложения двоичных чисел произвольной размерности.

```
.data
sl1     dd      01fe544fh
sl2     dd      005044cdh
elderREZ        db      0 ;для учета
переноса из старшего разряда результата
rez     dd      0
.code
...
        mov     ax,sl1
        add     ax,sl2  ;сложение
младших слов слагаемых
        mov     rez,ax
        mov     ax,sl+2
        adc     ax,sl2+2
;сложение старших слов слагаемых плюс
cf
        mov     rez+2,ax
        adc     elderREZ,0      ;учесть
возможный перенос
```

См. также: урок 8, приложение 7 и команды add, sub, sbb, xadd

ADD

(ADDition)
Сложение

Схема команды: add приемник,источник
Назначение: сложение двух операндов источник и приемник размерностью байт, слово или двойное слово.

Синтаксис
Алгоритм работы:

- сложить операнды источник и приемник;
- записать результат сложения в приемник;
- установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команда add используется для сложения двух целочисленных операндов. Результат сложения помещается по адресу первого операнда. Если результат сложения выходит за границы операнда приемник (возникает переполнение), то учесть эту ситуацию следует путем анализа флага cf и последующего возможного применения команды adc. Например, сложим значения в регистре ax и области памяти ch. При сложении следует учесть возможность переполнения.

```
chiclo  dw      2015
rez     dd      0
...
        add     ax,chislo
;(ax)=(ax)+ch
        mov     word ptr rez,ax
        jnc     dop_sum
;переход, если результат не вышел за
разрядную сетку
        adc     word ptr rez+2,0
;расширить результат, для учета
переноса
                                ;в
старший разряд
dop_sum:
...
```

См. также: урок 8, Приложение 7 и команды adc, sub, sbb, xadd

AND

(logical AND)
Логическое И

Схема команды: and приемник,источник
Назначение: операция логического умножения для операндов приемник и источник размерностью байт, слово или двойное слово.

Синтаксис
Алгоритм работы:

- выполнить операцию логического умножения над операндами источник и приемник: каждый бит результата равен 1, если соответствующие биты операндов равны 1, в остальных случаях бит результата равен 0;
- записать результат операции в приемник;
- установить флаги.

Состояние флагов после выполнения команды:

11	07	06	02	00
OF	SF	ZF	PF	CF
0	r	r	r	0

Применение:
Команда and используется для логического умножения двух операндов. Результат операции помещается по адресу первого операнда. Эту команду удобно использовать для принудительной установки или сброса определенных битов операнда.
Например, преобразуем двузначное упакованное BCD-число в его символьный эквивалент.

```
u_BCD   db      25h ;упакованное BCD-
число
s_ch    dw      0 ;место для результата
...
        xor     ax,ax   ;очистка ax
        mov     al,u_BCD
        shl     ax,4    ;ax=0250
        mov     al,u_BCD
;ax=0225
;преобразование в символьное
представление:
        and     ax,3f3fh
;ax=3235h
        mov     s_ch,ax
```

См. также: уроки 9, 12 и команды or, xor, test

BOUND

(check array BOUNDs)
Контроль нахождения индекса массива в границах

Схема команды: bound индекс,границы массива
Назначение: проверка нахождения значения индекса в границах массива.

Синтаксис
Алгоритм работы:
Сравнить значение в регистре индекс с двумя значениями, расположенными последовательно в ячейке памяти, адресуемой операндом границы массива. Диапазон значений индекса определяется используемым регистром индекс:

- если это 16-разрядный регистр общего назначения, то содержащееся в нем значение проверяется на попадание в диапазон значений, которые находятся в двух последовательных словах в памяти по адресу, указываемому вторым операндом.

Эти два значения являются, соответственно, значениями нижнего и верхнего индекса границы массива;

- если это 32-разрядный регистр общего назначения, то содержащееся в нем значение проверяется на попадание в диапазон значений, которые находятся в двух последовательных двойных словах в памяти по адресу, указываемому вторым операндом. Эти два значения являются, соответственно, значениями нижнего и верхнего индекса границы массива;

Если в результате проверки значение из регистра вышло за пределы указанного диапазона значений, то возбуждается прерывание с номером 5, если нет, программа продолжает выполнение.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команду bound очень удобно использовать для контроля выхода за нижнюю или верхнюю границы массива. Значения этих границ должны быть предварительно помещены в два последовательных слова (двойных слова) в памяти. Адрес этих слов (двойных слов) указывается вторым операндом. Далее динамически в ходе работы программы значение в регистре индекс, указываемом первым операндом, сравнивается со значениями этих двух границ, и если нижняя_граница<=(индексindex)<=верхняя_граница, то программа продолжает выполнение. В противном случае генерируется исключительная ситуация 5 (int 5). Далее в программе обработки этой ситуации можно выполнить необходимую корректировку и вернуться в программу (см. урок 17).

Фрагмент, который можно использовать при обработке одномерного массива с размерностью элементов в слово:

```
.286    ;это обязательная директива,
так как bound
        ;входит в систему команд
микропроцессоров, начиная с i286
.data
BoundMas        label   word
Low_Bound       dw      0
Upp_Bound       dw      20
mas     dw      10 dup (?)
...
        xor     di,di   ;очистка
индексного регистра
cycl:
        mov     ax,mas[di]
;перебор
элементов массива
        add     di,2
        bound   di,BoundMas
;если значение в di не будет попадать в
границы, то будет вызван
;обработчик прерывания 5, где можно
скорректировать
;значение ip/eip в стеке с тем, чтобы
выйти
;из бесконечного ;цикла, например, на
метку M2 или
```

```
;выполнить другие действия
        jmp     cycl
M2:
...
```

См. также: урок 17 и команду iret/iretd

BSF

(Bit Scan Forward)
Побитное сканирование вперед

Схема команды: bsf результат,источник
Назначение: для проверки наличия единичных битов в операнде источник.

Синтаксис
Алгоритм работы:

- просмотр битов операнда источник, начиная с бита 0 и заканчивая битом 15/31, до тех пор, пока не встретится единичный бит;
- если встретился единичный бит, то флаг zf устанавливается в 0 и в регистр первого операнда записывается номер позиции, где встретился единичный бит. Диапазон значений зависит от разрядности второго операнда: для 16-разрядного операнда — это 0...15; для 32-разрядного — это 0...31;
- если единичных битов нет, то флаг zf устанавливается в 1.

Состояние флагов после выполнения команды:

06
ZF
r

Применение:
Команду bsf используют при работе на битном уровне для определения позиции в операнде крайних справа единичных битов.
Например, сдвинем содержимое регистра bx вправо таким образом, чтобы нулевой бит стал единичным:

```
.386
        mov     bx,0002h
;bx=0000 0010b
...
        bsf     cx,bx   ;cx=0001h
        jz      null
        shr     bx,cl   ;bx=0000 0001b
...
null:
```

См. также: урок 9, 12 и команду bsr

BSR

(Bit Scan Reverse)
Побитное сканирование назад

Схема команды: bsr результат,источник
Назначение: проверка наличия единичных битов в операнде источник.

Синтаксис
Алгоритм работы:

- просмотр битов операнда источник, начиная со старшего бита 15/31 и заканчивая битом 0, до тех пор, пока не встретится единичный бит;
- если встретился единичный бит, флаг zf устанавливается в 0 и в регистр первого операнда записывается номер позиции (отсчет осуществляется относительно нулевой позиции), где встретился самый старший единичный бит. Диапазон значений зависит от разрядности второго операнда: для 16-разрядного операнда это 0...15; для 32-разрядного — 0...31;
- если единичных битов нет, флаг zf устанавливается в 1.

Состояние флагов после выполнения команды:

Применение:
Команду bsr используют при работе на битном уровне для определения позиции крайних слева единичных битов.

Например, сдвинем содержимое регистра bx вправо таким образом, чтобы старший единичный бит исходного значения в bx переместился в нулевую позицию:

```
.386
        mov     bx,41h
...
        bsr     cx,bx   ;cx=06h
        jz      null
        shr     bx,ax   ;bx=0001h
...
null:...
```

См. также: уроки 9, 12 и команду bsf

BSWAP

(Byte SWAP)
Перестановка байтов

Схема команды: bswap источник
Назначение:

- изменение порядка следования байтов;
- переход от одной формы адресации к другой.

Под формой адресации здесь понимается принцип "младший байт по младшему адресу" или обратный ему. Существует ряд систем, например использующих микропроцессоры Motorola или большие ЭВМ, где применяется принцип размещения многобайтовых значений обратный тому, который используется в микропроцессорах Intel. Поэтому эту команду можно использовать для разработки программ-конверторов между подобными платформами и IBM PC.

Синтаксис
Алгоритм работы: *Схема алгоритма*
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команду bswap используют для изменения формы адресации. В качестве операнда может быть указан только 32-разрядный регистр. Эта команда используется в моделях микропроцессоров, начиная с i486.

```
.486
        mov     ebx,1a2c345fhh
        bswap   ebx     ;ebx=5f342c1ah
```

См. также: урок 7, и команду xchg

BT

(Bit Test)
Проверка битов

Схема команды: bt источник,индекс
Назначение: извлечение значения заданного бита в флаг cf.

Синтаксис
Алгоритм работы:

- получить бит по указанному номеру позиции в операнде источник;
- установить флаг cf согласно значению этого бита.

Состояние флагов после выполнения команды:

00
CF
r

Применение:
Команду bt используют для определения значения конкретного бита в операнде источник. Номер проверяемого бита задается содержимым второго операнда (значение числом из диапазона 0...31). После выполнения команды, флаг cf устанавливается в соответствии со значением проверяемого бита.

```
.386
        mov     ebx,01001100h
        bt      ebx,8   ;проверка
состояния бита 8 и установка cf= в 1
        jc      m1      ;перейти на m1,
если проверяемый бит равен 1
...
```

См. также: уроки 9, 12 и команды btc, btr, bts, test

BTC

(Bit Test and Complement)
Проверка бита с инверсией (дополнением)

Схема команды: btc источник,индекс

Назначение: извлечение значения заданного бита в флаг cf и изменение его значения в операнде на обратное.

Синтаксис

Алгоритм работы:

- получить значение бита с номером позиции индекс в операнде источник;
- инвертировать значение выбранного бита в операнде источник;
- установить флаг cf исходным значением бита.

Состояние флагов после выполнения команды:

00
CF
r

Применение:
Команда btc используется для определения и инвертирования значения конкретного бита в операнде источник. Номер проверяемого бита задается содержимым второго операнда индекс (значение из диапазона 0...31). После выполнения команды флаг cf устанавливается в соответствии с исходным значением бита, то есть тем, которое было до выполнения команды.

```
.386
        mov     ebx,01001100h
;проверка состояния бита 8 и его
обращение:
        btc     ebx,8   ;cf=1 и
ebx=01001000h
```

См. также: уроки 9, 12 и команды bt, btr, bts, test

BTR

(Bit Test and Reset)
Проверка бита с его сбросом в 0

Схема команды: btr источник,индекс
Назначение: извлечение значения заданного бита в флаг cf и изменение его значения на нулевое.

Синтаксис
Алгоритм работы:

- получить значение бита с указанным номером позиции в операнде источник;
- установить флаг cf значением выбранного бита;
- установить значение исходного бита в операнде в 0.

Состояние флагов после выполнения команды:

Применение:
Команда btr используется для определения значения конкретного бита в операнде источник и его сброса в 0. Номер проверяемого бита задается содержимым второго операнда индекс (значение из диапазона 0...31). В результате выполнения команды флаг cf устанавливается в соответствии со значением исходного бита, то есть тем, что было до выполнения операции.

```
.386
        mov     ebx,01001100h
;проверка состояния бита 8 и его сброс
в 0
        btr     ebx,8   ;cf=1 и
ebx=01001000h
```

См. также: уроки 9, 12 и команды bt, btc, bts, test

BTS

(Bit Test and Set)
Проверка бита с его установкой в 1

Схема команды: bts источник,индекс
Назначение: извлечение значения заданного бита операнда в флаг cf и установка этого бита в единицу.

Синтаксис
Алгоритм работы:

- получить значение бита с указанным номером позиции в операнде источник;
- установить флаг cf значением выбранного бита;
- установить значение исходного бита в операнде источник в 1.

Состояние флагов после выполнения команды:

Применение:
Команда bts используется для определения значения конкретного бита в операнде источник и установки проверяемого бита в 1. Номер проверяемого бита задается содержимым второго операнда индекс (значение из диапазона 0...31). После выполнения команды флаг cf устанавливается в соответствии со значением исходного бита, то есть тем, что было до выполнения операции.

```
.386
        mov     ebx,01001100h
;проверка состояния бита 0 и его
установка в 1
        bts     ebx,0   ;cf=0
ebx=01001001h
```

См. также: уроки 9, 12 и команды bt, btc, btr, test

CALL

(CALL)
Вызов процедуры или задачи

Схема команды: call цель
Назначение:

- передача управления близкой или дальней процедуре с запоминанием в стеке адреса точки возврата;
- переключение задач.

Синтаксис
Алгоритм работы:
определяется типом операнда:

- метка ближняя — в стек заносится содержимое указателя команд eip/ip и в этот же регистр загружается новое значение адреса, соответствующее метке;
- метка дальняя — в стек заносится содержимое указателя команд eip/ip и cs. Затем в эти же регистры загружаются новые значения адресов, соответствующие дальней метке;
- r16, 32 или m16, 32 — определяют регистр или ячейку памяти, содержащие смещения в текущем сегменте команд, куда передается управление. При передаче управления в стек заносится содержимое указателя команд eip/ip;
- указатель на память — определяет ячейку памяти, содержащую 4 или 6-байтный указатель на вызываемую процедуру. Структура такого указателя 2+2 или 2+4 байта. Интерпретация такого указателя зависит от режима работы микропроцессора:
- в реальном режиме — в зависимости от размера адреса (use16 или use32) первые два байта трактуются как сегментный адрес, вторые два/четыре байта, как

смещение целевой метки передачи управления. В стеке запоминается содержимое регистров cs и eip/ip;
- в защищенном режиме — интерпретация цели передачи управления зависит от значения байта AR дескриптора, определяемого селекторной частью указателя. Целью здесь являются дальний вызов процедуры без изменения уровня привилегий, дальний вызов процедуры с изменением уровня привилегий или переключение задачи.

Состояние флагов после выполнения команды (кроме переключения задачи):

```
выполнение команды не влияет на флаги
```

При *переключении задачи* значения флажков изменяются в соответствии с информацией о регистре eflags в сегменте состояния TSS задачи, на которую производится переключение.
Применение:
Как видно из описания алгоритма, команда call позволяет организовать гибкую и многовариантную передачу управления на подпрограмму с сохранением адреса точки возврата. Подробно типовые примеры использования рассмотрены на уроках 10 и 14.
См. также: уроки 10, 14 и команду *ret*

CBW/CWDE

(Convert Byte to Word/Convert Word to Double Word Extended)
Преобразование байта в слово/слова в двойное слово

Схема команды:

cbw
cwde

Назначение: расширение операнда со знаком.

Синтаксис
Алгоритм работы:
cbw — при работе команда использует только регистры al и ax:

- анализ знакового бита регистра al:
  - если знаковый бит al=0, то ah=00h;
  - если знаковый бит al=1, то ah=0ffh.

cwde — при работе команда использует только регистры ax и eax:

- анализ знакового бита регистра ax:
  - если знаковый бит ax=0, то установить старшее слово eax=0000h;
  - если знаковый бит ax=1, то установить старшее слово eax=0ffffh.

Состояние флагов после выполнения команды:

```
выполнение команды не влияет на флаги
```

Применение:
Данные команды используются для приведения операндов к нужной размерности с учетом знака. Такая необходимость может, в частности, возникнуть при программировании арифметических операций.

```
.386    ;только для cwde, cwd была для
```

```
i8086
        mov     ebx,10fecd23h
        mov     ax,-3   ;ax=1111 1111
1111 1101
        cwde    ;eax=1111 1111 1111
1111 1111 1111 1111 1101
        add     eax,ebx
```

См. также: урок 8 и команды cdq, cwd

CLC

(CLear Carry flag)
Сброс флага переноса

Схема команды: clc
Назначение: сброс флага переноса cf.

Синтаксис
Алгоритм работы:
установка флага cf в ноль.
Состояние флагов после выполнения команды:

00
CF
0

Применение:
Данная команда используется для сброса флага cf в ноль. Такая необходимость может возникнуть при работе с командами сдвига, арифметическими командами либо действиями по индикации обнаружения ошибок и различных ситуаций в программе.

```
        clc             ;cf=0
```

См. также: уроки 8, 9 и команды cmc, stc

CLD

(CLear Direction flag)
Сброс флага направления

Схема команды: cld
Назначение: сброс в ноль флага направления df.

Синтаксис
Алгоритм работы:
установка флага df в ноль.
Состояние флагов после выполнения команды:

10
DF
0

Применение:
Данная команда используется для сброса флага df в ноль. Такая необходимость может возникнуть при работе с цепочечными командами. Нулевое занчение флага df вынуждает микропроцессор при выполнении цепочечных операций производить инкремент регистров si и di.

```
        cld             ;df=0
```

См. также: урок 11 и команды stc, movs/movsb/movsw/movsd, cmps/cmpsb/cmpsw/cmpsd, scas/scasb/scasw/scasd, lods/lodsb/lodsw/lodsd, stos/stosb/stosw/stosd, ins/insb/insw/insd, outs

CLI

(CLear Interrupt flag)
Сброс флага прерывания

Схема команды: cli
Назначение: сброс флага прерывания if.

Синтаксис
Алгоритм работы:
установка флага if в ноль.
Состояние флагов после выполнения команды:

09
IF
0

Применение:
Данная команда используется для сброса флага if в ноль. Такая необходимость может возникнуть при разработке программ обработки прерываний.

```
        cli             ;if=0
```

См. также: урок 15 и команды int, iret/iretd, sti

CMC

(CoMplement Carry flag)
Инвертирование флага переноса

Схема команды: cmc
Назначение: изменение значения флага переноса cf на обратное.

Синтаксис
Алгоритм работы:
инвертирование значения флага переноса cf.
Состояние флагов после выполнения команды:

00
CF
r

Применение:
Данная команда используется для изменения значения флага cf на противоположное. В частности, этот флаг можно использовать для связи с процедурой и по его состоянию судить о результате работы данной процедуры. После выхода из процедуры этот флаг можно проанализировать командой условного перехода jc.

```
proc1   proc
...
        cmc
...
proc1   endp
...
        call    proc1
        jc      m1      ;если cf=1, то
переход на m1
...
m1:
...
```

См. также: уроки 8, 9, 15 и команды clc, stc, jc, jnc

CMP

(CoMPare operands)
Сравнение операндов

Схема команды: cmp операнд1,операнд2
Назначение: сравнение двух операндов.

Синтаксис
Алгоритм работы:

- выполнить вычитание (операнд1-операнд2);
- в зависимости от результата установить флаги, операнд1 и операнд2 не изменять (то есть результат не запоминать).

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Данная команда используется для сравнения двух операндов методом вычитания, при

этом операнды не изменяются. По результатам выполнения команды устанавливаются флаги. Команда cmp применяется с командами условного перехода и командой установки байта по значению setcc.

```
len     equ     10
...
        cmp     ax,len
        jne     m1      ;переход если
(ax)<>len
        jmp     m2      ;переход если
(ax)=len
```

См. также: уроки 10, 11, 12 и команды cmps/cmpsb/cmpsw/cmpsd, cmpxchg, sub, jcc, setcc

CMPS/CMPSB/CMPSW/CMPSD

(CoMPare String Byte/Word/Double word operands)
Сравнение строк байтов/слов/двойных слов

Схема команды:

cmps приемник,источник
cmpsb
cmpsw
cmpsd

Назначение: сравнение двух последовательностей (цепочек) элементов в памяти.

Синтаксис
Алгоритм работы:

- выполнить вычитание элементов (источник - приемник), адреса элементов предварительно должны быть загружены:
  - адрес источника — в пару регистров *ds*:esi/si;
  - адрес назначения — в пару регистров es:edi/di;
- в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:
  - если df=0, то увеличить содержимое этих регистров на длину элемента последовательности;
  - если df=1, то уменьшить содержимое этих регистров на длину элемента последовательности;
- в зависимости от результата вычитания установить флаги:
  - если очередные элементы цепочек не равны, то cf=1, zf=0;
  - если очередные элементы цепочек или цепочки в целом равны, то cf=0, zf=1;
- при наличии префикса выполнить определяемые им действия (см. команды repe/repne).

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команды без префиксов осуществляют простое сравнение двух элементов в памяти. Размеры сравниваемых элементов зависят от применяемой команды. Команда cmps может работать с элементами размером в байт, слово, двойное слово. В качестве операндов в

команде указываются идентификаторы последовательностей этих элементов в памяти. Реально эти идентификаторы используются лишь для получения типов элементов последовательностей, а их адреса должны быть предварительно загружены в указанные выше пары регистров. Транслятор, обработав команду cmps и выяснив тип операндов, генерирует одну из машинных команд cmpsb, cmpsw или cmpsd. Машинного аналога для команды cmps нет. Для адресации назначения обязательно должен использоваться регистр es, а для адресации источника можно делать замену сегмента с использованием соответствующего префикса.
Для того чтобы эти команды можно было использовать для сравнения последовательности элементов, имеющих размерность байт, слово, двойное слово, необходимо использовать один из префиксов repe или repne. Префикс repe заставляет циклически выполняться команды сравнения до тех пор, пока содержимое регистра *ecx/cx* не станет равным нулю или пока не совпадут очередные сравниваемые элементы цепочек (флаг zf=1). Префикс repne заставляет циклически производить сравнение до тех пор, пока не будет достигнут конец цепочки (*ecx/cx*=0) либо не встретятся различающиеся элементы цепочек (флаг zf=0).

```
.data
obl1    db      'Строка для сравнения'
obl1    db      'Строка для сравнения'
a_obl1  dd      obl1
a_obl2  dd      obl2
.code
...
        cld             ;просмотр
цепочки в направлении возрастания
адресов
        mov     cx,20   ;длина цепочки
        lds     si,a_obl1       ;адрес
источника в пару ds:si
        les     di,a_obl2       ;адрес
назначения в пару ds:si
repe    cmpsb           ;сравнивать,
пока равны
        jnz     m1      ;если не конец
цепочки, то встретились разные элементы
...                     ;действия, если
цепочки совпали
...
m1:
...                     ;действия, если
цепочки не совпали
```

См. также: уроки 10, 11 и команды *ins*, *lods*, *movs*, *outs*, *scas*, *stos*, *repe*, *repz*, *repne*, *repnz*

CMPXCHG

(CoMPare and eXCHanGe)
Сравнение и обмен

Схема команды:

cmpxchg
приемник,источник(аккумулятор)

Назначение: сравнение и обмен значений между источником и приемником.

Синтаксис
Алгоритм работы:

- выполнить сравнение элементов источник и приемник;
- если источник и приемник не равны, то:
  - установить zf=0;
  - переслать содержимое операнда приемник в источник (регистр al/ax/eax).
- если источник и приемник равны, то:
  - установить zf=1;
  - переслать содержимое операнда источник (регистр al/ax/eax) по месту операнда приемник.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команды сравнивают два операнда. Один из сравниваемых операндов находится в аккумуляторе (регистре al/ax/eax), другой может находиться в памяти или регистре общего назначения. Если значения равны, то производится замена содержимого операнда приемник содержимым источника, находящимся в регистре-аккумуляторе. Если значения не равны, то производится замена содержимого операнда источника находящимся в регистре-аккумуляторе содержимым операнда назначения. Определить тот факт, была ли произведена смена значения в аккумуляторе (то есть были ли не равны сравниваемые операнды), можно по значению флага zf.

```
.486
        mov     ax,114eh
        mov     bx,8e70h
        cmpxchg bx,ax
        jz      m1      ;переход, если
zf=1, то есть операнды равны
                        ;и ax не
изменился
...                     ;действия, если
операнды не равны
m1:
```

См. также: уроки 7, 10 и команды *cmp*, *xchg*

CWD

(Convert Word to Double word)
Преобразование слова в двойное слово

Схема команды: cwd
Назначение: расширение слова со знаком до размера двойного слова со знаком.

Синтаксис
Алгоритм работы:
копирование значения старшего бита регистра ax во все биты регистра dx. *Состояние флагов после выполнения команды:*

выполнение команды не влияет на флаги

Применение:
Команда cwd используется для расширения значения знакового бита в регистре ax на биты регистра dx. Данную операцию, в частности, можно использовать для подготовки к операции деления, для которой размер делимого должен быть в два раза больше размера делителя, либо для приведения операндов к одной размерности в командах умножения, сложения, вычитания.

```
        mov     ax,25
...
        mov     bx,4
        cwd
        div     bx
```

См. также: урок 8 и команды cbw, cdq, cwde, div, idiv, mul, imul, add, adc, sub, sbb

CDQ

(Convert Double word to Quad word)
Преобразование двойного слова в учетверенное слово

Схема команды: cdq
Назначение: расширение двойного слова со знаком до размера учетверенного слова (64 бита) со знаком.

Синтаксис
Алгоритм работы:
копирование значения старшего бита регистра eax на все биты регистра edx. *Состояние флагов после выполнения команды:*

выполнение команды не влияет на флаги

Применение:
Команду cdq можно использовать для распространения значения знакового бита в регистре eax на все биты регистра edx. Данную операцию, в частности, можно использовать для подготовки к операции деления, для которой размер делимого должен быть в два раза больше размера делителя.

```
.386
delimoe dd      ...
delitel dd      ...
...
        mov     eax,delimoe
        cdq
        idiv    delitel ;частное в eax,
остаток в edx
```

См. также: урок 8 и команды cbw, cwd, cwde, div, idiv

DAA

(Decimal Adjust for Addition)

Десятичная коррекция после сложения

Схема команды: daa

Назначение: коррекция упакованного результата сложения двух BCD-чисел в упакованном формате.

Синтаксис

Алгоритм работы:

команда работает только с регистром al и анализирует наличие следующих ситуаций:

- Ситуация 1. В результате предыдущей команды сложения флаг af=1 или значение младшей тетрады регистра al>9. Напомним, что флаг af устанавливается в 1 в случае переноса двоичной единицы из бита 3 младшей тетрады в старшую тетраду регистра al (если значение превысило 0fh). Наличие одного из этих двух признаков говорит о том, что значение младшей тетрады превысило 9h.
- Ситуация 2. В результате предыдущей команды сложения флаг cf=1 или значение регистра al>9fh. Напомним, что флаг cf устанавливается в 1 в случае переноса двоичной единицы в старший бит операнда (если значение превысило 0ffh в случае регистра al). Наличие одного из этих двух признаков говорит о том, что значение в регистре al превысило 9fh.

Если имеет место одна из этих двух ситуаций, то регистр al корректируется следующим образом:

- для ситуации 1 содержимое регистра al увеличивается на 6;
- для ситуации 2 содержимое регистра al увеличивается на 60h;
- если имеют место обе ситуации, то корректировка начинается с младшей тетрады.

Состояние флагов после выполнения команды (в случае, если были переносы):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	1	r	1

Состояние флагов после выполнения команды (в случае, если переносов не было):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	0	r	0

Применение:

Эту команду следует применять после сложения двух упакованных BCD-чисел с целью корректировки получающегося двоичного результата сложения в правильное двузначное десятичное число. После команды daa следует анализировать состояние флага cf. Если он равен 1, то это говорит о том, что был перенос единицы в старший разряд и это нужно учесть для сложения старших десятичных цифр BCD-числа.

```
        mov     al,69h  ;69h –
упакованное BCD-число
        mov     bl,74h  ;74h –
упакованное BCD-число
        adc     al,bl   ;al=0ddh
        daa             ;cf=1, al=43h
```

```
;если перенос, то переход на ту ветвь
программы,
;где он будет учтен:
        jc m1
```

См. также: урок 8, Приложение 7 и команды aaa, aad, aam, aas, das

DAS

(Decimal Adjust for Subtraction)
Десятичная коррекция после вычитания

Схема команды: das

Назначение: коррекция упакованного результата вычитания двух BCD-чисел в упакованном формате.

Синтаксис

Алгоритм работы:

команда das работает только с регистром al и анализирует наличие следующих ситуаций:

- Ситуация 1. В результате предыдущей команды сложения флаг af =1 или значение младшей тетрады регистра al>9. Напомним, что для случая вычитания флаг af устанавливается в 1 в случае заема двоичной единицы из старшей тетрады в младшую тетраду регистра al. Наличие одного из этих двух признаков говорит о том, что значение младшей тетрады превысило 9h и его нужно корректировать.
- Ситуация 2. В результате предыдущей команды сложения флаг cf =1 или значение регистра al>9fh. Напомним, что для случая вычитания флаг cf устанавливается в 1 в случае заема двоичной единицы. Наличие одного из этих двух признаков говорит о том, что значение в регистре al превысило 9fh.

Если имеет место одна из этих ситуаций, то регистр al корректируется следующим образом:

- для ситуации 1 содержимое регистра al уменьшается на 6;
- для ситуации 2 содержимое регистра al уменьшается на 60h;
- если имеют место обе ситуации, то корректировка начинается с младшей тетрады.

Состояние флагов после выполнения команды (в случае, если были переносы):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	1	r	1

Состояние флагов после выполнения команды (в случае, если переносов не было):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	0	r	0

Применение:

Команду das следует применять после вычитания двух упакованных BCD-чисел с целью

корректировки получающегося двоичного результата вычитания в правильное двузначное десятичное число. После команды das следует анализировать состояние флага cf. Если он равен 1, то это говорит о том, что был заем единицы в старший разряд и это нужно учесть в дальнейших действиях. Если у вычитаемого нет больше старших разрядов, то результат следует трактовать как отрицательное двоичное дополнение. Для определения его абсолютного значения нужно вычесть 100 из результата в al. Если у вычитаемого еще есть старшие разряды, то факт заема нужно просто учесть уменьшением младшего из этих оставшихся старших разрядов на единицу.

```
        mov     ah,08h  ;ah=08h
        mov     al,05h  ;al=05h
        add     al,ah
;al=al+ah=05h+08h=0dh – не BCD-число
        xor     ah,ah   ;ah=0
        aaa             ;ah=01h,al=03h
– результат скорректирован
```

См. также: урок 8, Приложение 7 и команды aaa, aad, aam, aas, daa

DEC

(DECrement operand by 1)
Уменьшение операнда на единицу

Схема команды: dec операнд
Назначение: уменьшение значения операнда в памяти или регистре на 1.

Синтаксис
Алгоритм работы:
команда вычитает 1 из операнда. *Состояние флагов после выполнения команды:*

11	07	06	04	02
OF	SF	ZF	AF	PF
r	r	r	r	r

Применение:
Команда dec используется для уменьшения значения байта, слова, двойного слова в памяти или регистре на единицу. При этом заметьте то, что команда не воздействует на флаг cf.

```
        mov     al,9
...
        dec     al      ;al=8
```

См. также: урок 8 и команды *inc*, *sub*

DIV

(DIVide unsigned)
Деление беззнаковое

Схема команды: div делитель

Назначение: выполнение операции деления двух двоичных беззнаковых значений.

Синтаксис

Алгоритм работы:

Для команды необходимо задание двух операндов — делимого и делителя. Делимое задается неявно и размер его зависит от размера делителя, который указывается в команде:

- если делитель размером в байт, то делимое должно быть расположено в регистре ax. После операции частное помещается в al, а остаток — в ah;
- если делитель размером в слово, то делимое должно быть расположено в паре регистров dx:ax, причем младшая часть делимого находится в ax. После операции частное помещается в ax, а остаток — в dx;
- если делитель размером в двойное слово, то делимое должно быть расположено в паре регистров edx:eax, причем младшая часть делимого находится в eax. После операции частное помещается в eax, а остаток — в edx.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	?	?	?	?	?

Применение:

Команда выполняет целочисленное деление операндов с выдачей результата деления в виде частного и остатка от деления. При выполнении операции деления возможно возникновение исключительной ситуации: 0 — ошибка деления. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре eax/ax/al.

```
        mov     ax,10234
        mov     bl,154
        div     bl      ;ah=остаток,
al=частное
```

См. также: урок 8, приложение 7 и команду idiv

ENTER

(setup parameter block for ENTERing procedure)
Установка кадра стека для параметров процедуры

Схема команды: enter loc_size,lex_lev

Назначение: установка границы в стеке для локальных переменных процедуры.

Синтаксис

Алгоритм работы:

- поместить текущее значение регистра ebp/bp в стек;
- сохранить текущее значение esp/sp в промежуточной переменной fp (имя переменной выбрано случайно);

- если лексический уровень вложенности (операнд lex_lev) не равен нулю, то (lex_lev–1) раз делать следующее:
    - в зависимости от установленного режима адресации use16 или use32 выполнить вычитание (bp–2) или (ebp–4) и записать результат обратно в ebp/bp;
    - сохранить значение ebp/bp в стеке;
    - сохранить в стеке значение промежуточной переменной fp;
- записать значение промежуточной переменной fp в регистр ebp/bp;
- уменьшить значение регистра esp/sp на величину, заданную первым операндом, минус размер области локальных переменных loc_size: esp/sp=(esp/sp)–loc_size.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда enter специально введена в систему команд микропроцессора для поддержки блочно-структурированных языков высокого уровня типа Pascal или C. В этих языках программа разбивается на блоки. В блоках можно описать свои собственные (локальные) идентификаторы, которые не могут быть использованы вне этого блока. К примеру, на рисунке ниже в виде блоков изображена структура некоторой программы.

Изображение структуры некоторой программы в виде блоков

В правом верхнем углу каждого блока (процедуры) стоит номер лексического уровня вложенности этого блока относительно других блоков программы. Большинство блочно-структурированных языков в качестве основного метода распределения памяти для переменных в блоках используют автоматическое распределение памяти. Это означает, что при входе в блок (вызове процедуры и т. п.) в некотором месте памяти (или в стеке) выделяется область памяти для переменных этого блока (ее можно назвать областью инициализации). После выхода из этого блока связь программы с этой областью теряется, то есть эти переменные становятся недоступными. Но если, как в нашем примере, в этой процедуре есть вложенные блоки (процедуры), то для некоторого внутреннего блока

(например, C) могут быть доступны области инициализации (переменные) блоков, объемлющих данный блок. В нашем примере для блока C доступны также переменные блоков B и A, но не D. Возникает вопрос: как же программа, находясь в конкретной точке своего выполнения, может отслеживать то, какие области инициализации ей доступны? Это делается с помощью структуры данных, называемой дисплеем. Дисплей содержит указатели на самую последнюю область текущего блока и на области инициализации всех блоков, объемлющих данный блок в программе. Например, если в программе A была вызвана сначала процедура B, а затем C, то дисплей содержит указатели на области инициализации A, B и C (см. рисунок ниже).

Если после этого вызвать процедуру D (в то время как B и C еще не завершены), то картина изменится.

После того как некоторый блок (процедура) завершает свою работу, ее область инициализации удаляется из памяти (стека) и одновременно соответствующим образом корректируется дисплей. Большинство языков высокого уровня хранят локальные данные блоков в стеке. Эти переменные называют еще автоматическими или динамическими. Память для них резервируется путем уменьшения значения регистра-указателя стека esp/sp на величину, равную длине области, занимаемой этими динамическими переменными. Доступ к этим переменным осуществляется посредством регистра ebp/bp. Если один блок вложен в другой, то для его динамических (локальных) переменных также выделяется место (кадр) в стеке, но в этот кадр помещается указатель на кадр стека для включающего его блока. Команды enter и leave как раз и позволяют поддержать в языке ассемблера принципы работы с переменными блоков как в блочно-структурированных языках. Дисплей организуется с помощью второго операнда команды enter и стека. Например, в начале работы главной процедуры A и после вызова процедуры B кадр стека будет выглядеть так.

Соответственно, после вызова процедур C и D стек будет выглядеть, как показано ниже.

Таким образом, видно, что используя дисплей, мы фактически имеем адреса областей инициализации, доступных по признаку вложенности объемлющих блоков. Обратный процесс завершения работы с блоками и удаления соответствующих областей инициализации поддерживается командой leave.

```
.286
proc1   proc
;зарезервировать в стеке место для
локальных переменных
;proc1 16 байт
;лексический уровень вложенности 0
        enter   16,0
...
        leave
        ret
proc1   endp
```

См. также: урок 14 и команды leave, *ret*

HLT

(HaLT)
Остановка

Схема команды: hlt
Назначение: остановка микропроцессора до прерывания или перезагрузки.

Синтаксис
Алгоритм работы:
перевод микропроцессора в состояние остановки.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
В результате выполнения команды микропроцессор переходит в состояние остановки. Из этого состояния его можно вывести сигналами на входах RESET, NMI, INTR. Если для возобновления работы микропроцессора используется прерывание, то сохраненное значение пары cs:eip/ip указывает на команду, следующую за hlt. Для иллюстрации применения данной команды рассмотрим еще один способ переключения микропроцессора из защищенного в реальный режим и его возврата обратно в реальный режим (см. урок 16). Как известно, в микропроцессоре не предусмотрено специальных средств для подобного переключения. Сброс микропроцессора можно инициировать, если вывести байт со значением 0feh в порт клавиатуры 64h. После этого микропроцесор переходит в реальный режим и управление получает программа BIOS, которая анализирует байт отключения в CMOS-памяти по адресу 0fh. Для нас интерес представляют два значения этого байта — 5h и 0ah:

- 5h — сброс микропроцессора инициирует инициализацию программируемого контроллера прерываний на значение базового вектора 08h (см. уроки 15 и 17). Далее управление передается по адресу, который находится в ячейке области данных BIOS 0040:0067;
- 0ah — сброс микропроцессора инициирует непосредственно передачу управления по адресу в ячейке области данных BIOS 0040:0067 (то есть без перепрограммирования контроллера прерываний).

Таким образом, если вы не используете прерываний, то достаточно установить байт 0fh в CMOS-памяти в 0ah. Предварительно, конечно, вы должны инициализировать ячейку области данных BIOS 0040:0067 значением адреса, по которому необходимо передать управление после сброса. Для программирования CMOS-памяти используются номера портов 070h и 071h. Вначале в порт 070h заносится нужный номер ячейки CMOS-памяти, а затем в порт 071h — новое значение этой ячейки.

```
;работаем в реальном режиме, готовимся
к переходу
;в защищенный режим:
        push    es
        mov     ax,40h
        mov     es,ax
        mov     word ptr
es:[67h],offset ret_real
```

```
;ret_real — метка в программе, с
которой должно
;начаться выполнение программы после
сброса
        mov     es:[69h],cs
        mov     al,0fh  ;будем
обращаться к ячейке 0fh в CMOS
        out     70h,al
        jmp     $+2     ;чуть
задержимся, чтобы аппаратура отработала
;сброс без перепрограммирования
контроллера
        mov     al,0ah
        out     71h,al
;переходим в защищенный режим
установкой
;бита 0 cr0 в 1 (см. урок 16)
;работаем в защищенном режиме
;готовимся перейти обратно в реальный
режим
        mov     al,01fch
        out     64h,al  ;сброс
микропроцессора  hlt
;остановка до физического окончания
процесса сброса
        ret_real:       ...     ;метка,
на которую будет передано
                        ;управление
после сброса
```

См. также: уроки 15, 16, 17

IDIV

(Integer DIVide)
Деление целочисленное со знаком

Схема команды: idiv делитель

Назначение: операция деления двух двоичных значений со знаком.

Синтаксис
Алгоритм работы:
Для команды необходимо задание двух операндов — делимого и делителя. Делимое задается неявно, и размер его зависит от размера делителя, местонахождение которого указывается в команде:

- если делитель размером в байт, то делимое должно быть расположено в регистре ax. После операции частное помещается в al, а остаток — в ah;
- если делитель размером в слово, то делимое должно быть расположено в паре регистров dx:ax, причем младшая часть делимого находится в ax. После операции частное помещается в ax, а остаток — в dx;
- если делитель размером в двойное слово, то делимое должно быть расположено в паре регистров edx:eax, причем младшая часть делимого находится в eax. После операции частное помещается в eax, а остаток — в edx;

Остаток всегда имеет знак делимого. Знак частного зависит от состояния знаковых битов (старших разрядов) делимого и делителя.
Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	?	?	?	?	?

Применение:
Команда выполняет целочисленное деление операндов с учетом их знаковых разрядов. Результатом деления являются частное и остаток от деления. При выполнении операции деления возможно возникновение исключительной ситуации: 0 — ошибка деления. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре eax/ax/al.

```
;деление слов
        mov     ax,1045 ;делимое
        mov     bx,587  ;делитель
        cwd             ;расширение
делимого dx:ax
        idiv    bx      ;частное в ax,
остаток в dx
```

См. также: урок 8, приложение 7 и команду div

IMUL

(Integer MULtiply)
Умножение целочисленное со знаком

Схема команды:
imul множитель_1
imul множ_1,множ_2
imul рез-т,множ_1,множ_2

Назначение: операция умножения двух целочисленных двоичных значений со знаком.

Синтаксис
Алгоритм работы:
Алгоритм работы команды зависит от используемой формы команды. Форма команды с одним операндом требует явного указания местоположения только одного сомножителя, который может быть расположен в ячейке памяти или регистре. Местоположение второго сомножителя фиксировано и зависит от размера первого сомножителя:

- если операнд, указанный в команде, — байт, то второй сомножитель располагается в al;
- если операнд, указанный в команде, — слово, то второй сомножитель располагается в ax;
- если операнд, указанный в команде, — двойное слово, то второй сомножитель располагается в eax.

Результат умножения для команды с одним операндом также помещается в строго определенное место, определяемое размером сомножителей:

- при умножении байтов результат помещается в ax;

- при умножении слов результат помещается в пару dx:ax;
- при умножении двойных слов результат помещается в пару edx:eax.

Команды с двумя и тремя операндами однозначно определяют расположение результата и сомножителей следующим образом:

- в команде с двумя операндами первый операнд определяет местоположение первого сомножителя. На его место впоследствии будет записан результат. Второй операнд определяет местоположение второго сомножителя;
- в команде с тремя операндами первый операнд определяет местоположение результата, второй операнд — местоположение первого сомножителя, третий операнд может быть непосредственно заданным значением размером в байт, слово или двойное слово.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	?	?	?	?	r

Команда imul устанавливает в ноль флаги of и cf, если размер результата соответствует регистру назначения. Если эти флаги отличны от нуля, то это означает, что результат слишком велик для отведенных ему регистром назначения рамок и необходимо указать больший по размеру регистр для успешного завершения данной операции умножения. Конкретными условиями сброса флагов of и cf в ноль являются следующие условия:

- для одноперандной формы команды imul регистры ax/dx/edx являются знаковыми расширениями регистров al/ax/eax;
- для двухоперандной формы команды imul для размещения результата умножения достаточно размерности указанных регистров назначения r16/r32;
- то же для трехоперандной команды умножения.

Применение:
Команда выполняет целочисленное умножение операндов с учетом их знаковых разрядов. Для выполнения этой операции необходимо наличие двух сомножителей. Размещение и задание их местоположения в команде зависит от формы применяемой команды умножения, которая, в свою очередь, определяется моделью микропроцессора. Так, для микропроцессора i8086 возможна только одноперандная форма команды, для последующих моделей микропроцессоров дополнительно можно использовать двух- и трехоперандные формы этой команды.

```
.486
...
        mov     bx,186
        imul    eax,bx,8
;если результату не хватило размерности
операнда1,
;то перейдем на m1, где скорректируем
ситуацию:
        jc      m1
```

См. также: урок 8, приложение 7 и команду mul

IN

(INput operand from port)
Ввод операнда из порта

Схема команды: in аккумулятор,ном_порта
Назначение: ввод значения из порта ввода-вывода.

Синтаксис
Алгоритм работы:
Передает байт, слово, двойное слово из порта ввода-вывода в один из регистров al/ax/eax.
Состояние флагов после выполнения команды: выполнение команды не влияет на флаги.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда применяется для прямого управления оборудованием компьютера посредством портов. Номер порта задается вторым операндом в виде непосредственного значения или значения в регистре dx. Непосредственным значением можно задать порт с номером в диапазоне 0-255. При использовании порта с большим номером используется регистр dx. Размер данных определяется размерностью первого операнда и может быть байтом, словом, двойным словом. В качестве примера применения рассмотрим фрагмент обработчика прерывания от клавиатуры 9. Это прерывание вызывается всякий раз при нажатии любой клавиши на клавиатуре. Обработчик этого прерывания должен прочитать скан-код клавиши, подтвердить микропроцессору клавиатуры факт приема скан-кода, преобразовать этот код в соответствии с клавишами-переключателями и поместить преобразованный код в буфер клавиатуры, находящийся в области BIOS. Действия чтения и подтверждения приема скан-кода могут выглядеть, к примеру, так:

```
        in      al,60h  ;читаем скан-
код
        push    ax      ;сохраним его
на время
        in      al,61h  ;читаем порт
61h
        or      al,80h  ;старший бит
байта из порта 61h в 1
        out     61h,al  ;подтверждаем
факт приема скан-кода
        pop     ax
        out     61h,al  ;восстановили
байт в порту 61h
```

См. также: урок 7 и команды out, ins/insb/insw/insd, outs

INC

(INCrement operand by 1)
Увеличить операнд на 1

Схема команды: inc операнд
Назначение: увеличение значения операнда в памяти или регистре на 1.

Синтаксис
Алгоритм работы:
команда увеличивает операнд на единицу.
Состояние флагов после выполнения команды:

11	07	06	04	02
OF	SF	ZF	AF	PF
r	r	r	r	r

Применение:
Команда используется для увеличения значения байта, слова, двойного слова в памяти или регистре на единицу. При этом команда не воздействует на флаг cf.

```
        inc     ax      ;увеличить
значение в ax на 1
```

См. также: урок 8 и команды dec, add, adc

INS/INSB/INSW/INSD

(Input String Byte/Word/Double word operands)
Ввод строк байтов/слов/двойных слов из порта

Схема команды:

ins приемник,порт
insb
insw
insd

Назначение: ввод из порта в память последовательности байт, слов, двойных слов.

Синтаксис
Алгоритм работы:

- передать данные из порта ввода-вывода, номер которого загружен в регистр dx, в память по адресу es:edi/di;
- в зависимости от состояния флага df изменить значение регистров edi/di:
  - если df=0, то увеличить содержимое этих регистров на длину структурного элемента последовательности;
  - если df=1, то уменьшить содержимое этих регистров на длину структурного элемента последовательности;
- при наличии префикса выполнить определяемые им действия (см. команду rep).

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда вводит данные из порта ввода-вывода, номер которого загружен в регистр dx, в память по адресу es:edi/di. Сегментная составляющая адреса должна быть обязательно в регистре es. Замена сегментного регистра недопустима. Непосредственное задание порта в команде также недопустимо - для этого используется регистр dx. Размеры вводимых элементов зависят от применяемой команды. Команда ins может работать с элементами размером в байт, слово, двойное слово. В качестве операндов в команде указывается

символическое имя ячейки памяти, в которую вводятся элементы из порта ввода-вывода. Реально это символическое имя используется лишь для получения типа элемента последовательности, а его адрес должен быть предварительно загружен в пару регистров es:edi/di. Транслятор, обработав команду ins и выяснив тип операнда, генерирует одну из машинных команд insb, insw или insd. Машинного аналога для команды ins нет. Для того чтобы эти команды можно было использовать для ввода последовательности элементов, имеющих размерность байт, слово, двойное слово, необходимо использовать префикс rep. Префикс rep заставляет циклически выполняться команду ввода до тех пор, пока содержимое регистра *ecx/cx* не станет равным нулю.

```
.286
;ввести 10 байт из порта 300h (номер
порта bgr условно)
;в цепочку байт в памяти по адресу
str_10  db      10 dup(0)
adr_str dd      str_10
        les     di,adr_str
        mov     dx,300h
rep     insb
...
```

См. также: уроки 2, 11 и команды cmps/cmpsb/cmpsw/cmpsd, lods/lodsb/lodsw/lodsd, movs/movsb/movsw/movsd, outs, scas/scasb/scasw/scasd, stos/stosb/stosw/stosd, rep/repe/repz/repne/repnz

INT

(INTerrupt)
Вызов подпрограммы обслуживания прерывания

Схема команды: int номер_прерывания

Назначение: вызов подпрограммы обслуживания прерывания с номером прерывания, заданным операндом команды.

Синтаксис
Алгоритм работы:

- записать в стек регистр флагов eflags/flags и адрес возврата. При записи адреса возврата вначале записывается содержимое сегментного регистра cs, затем содержимое указателя команд eip/ip;
- сбросить в ноль флаги if и tf;
- передать управление на программу обработки прерывания с указанным номером. Механизм передачи управления зависит от режима работы микропроцессора (см. уроки 15 и 17).

Состояние флагов после выполнения команды:

09	08
IF	TF
0	0

Применение:
Как видно из синтаксиса, существуют две формы этой команды:

- int 3 — имеет свой индивидуальный код операции 0cch и занимает один байт. Это обстоятельство делает ее очень удобной для использования в различных программных отладчиках для установки точек прерывания путем подмены первого байта любой команды. Микропроцессор, встречая в последовательности команд команду с кодом операции 0cch, вызывает программу обработки прерывания с номером вектора 3, которая служит для связи с программным отладчиком.
- Вторая форма команды занимает два байта, имеет код операции 0cdh и позволяет инициировать вызов подпрограммы обработки прерывания с номером вектора в диапазоне 0–255. Особенности передачи управления, как было отмечено, зависят от режима работы микропроцессора.

```
;вызов обработчика аппаратного
прерывания 08h из программы:
        int     08h
```

См. также: уроки 15, 17 и команды into, iret/iretd

INTO

(INTerrupt if Overflow)
Прерывание, если переполнение

Схема команды: into
Назначение: инициирование прерывания с номером 4, если установлен флаг of.

Синтаксис
Алгоритм работы:
Проанализировать состояние флага of:

- если of=0, то никаких действий производить не нужно — передать управление на следующую команду;
- если of=1, то дальнейшие действия, как при команде int, то есть:
  - записать в стек регистр флагов eflags/flags и адрес возврата. При записи адреса возврата вначале записывается содержимое сегментного регистра cs, затем содержимое указателя команд eip/ip;
  - сбросить в ноль флаги if и tf;
  - передать управление на программу обработки прерывания с данным номером. Механизм передачи зависит от режима работы микропроцессора (см. уроки 15 и 17).

Состояние флагов после выполнения команды:

09	08
IF	TF
r	r

Применение:
Свойство этой команды инициировать вызов подпрограммы обработки прерывания с номером вектора 4 определяет варианты ее применения. Если предыдущая команда в программе может в результате своей работы установить флаг переполнения of (к примеру,

арифметические команды), то для обнаружения и обработки такой ситуации можно использовать команду into. Особенности передачи управления и обработки (корректировки) результата зависят от режима работы микропроцессора.

```
.486
...
        mov     bx,186
        imul    eax,bx,8
;если результату не хватило размерности
операнда1,
;то of установится в 1
;исправим ситуацию в обработчике
прерывания 3
        into
```

См. также: уроки 8, 15, 17 и команды int, iret/iretd, imul

IRET/IRETD

(Interrupt RETurn)
Возврат из прерывания

Схема команды:

iret
iretd

Назначение: используется в той точке программы обработки прерывания, откуда необходимо вернуть управление прерванной программе.

Синтаксис
Алгоритм работы:
Работа команды зависит от режима работы микропроцесссора:

- в реальном режиме команда iret последовательно извлекает из стека и затем восстанавливает в микропроцессоре содержимое следующих регистров: eip/ip, cs, eflags/flags. Далее прерванная программа продолжается с точки прерывания;
- в защищенном режиме действия команды зависят от состояния флага NT (вложенной задачи) в регистре флагов:
  - если NT=0, то производятся действия по возврату управления прерванной программе, при этом характер этих действий зависит от соотношения уровней привилегированности прерванной программы и программы обработки прерывания;
  - в случае NT=1 производятся действия по переключению задач.

Состояние флагов после выполнения команды:

11	10	09	08	07	06	04	02	00
OF	DF	IF	TF	SF	ZF	AF	PF	CF
r	r	r	r	r	r	r	r	r

Применение:
Команду iret необходимо применять для восстановления сохраненных командой int регистров флагов, указателя команд и сегментного регистра кода. Число этих команд в программе обработки прерывания должно соответствовать количеству точек выхода из

нее. Команда iretd используется в старших моделях микропроцессоров для извлечения из стека и восстановления 32-битных регистров.

```
my_int1c        proc
;программа обработки прерывания 1Ch
...
        iret
        endp
```

См. также: уроки 15, 17 и команды int, into

JCC
JCXZ/JECXZ

(Jump if condition)
(Jump if CX=Zero/ Jump if ECX=Zero)
Переход, если выполнено условие
Переход, если CX/ECX равен нулю

Схема команды:
jcc метка
jcxz метка
jecxz метка

Назначение: переход внутри текущего сегмента команд в зависимости от некоторого условия.

Синтаксис
Алгоритм работы команд (кроме jcxz/jecxz):
Проверка состояния флагов в зависимости от кода операции (оно отражает проверяемое условие):

- если проверяемое условие истинно, то перейти к ячейке, обозначенной операндом;
- если проверяемое условие ложно, то передать управление следующей команде.

Алгоритм работы команды jcxz/jecxz:
Проверка условия равенства нулю содержимого регистра *ecx/cx*:

- если проверяемое условие истинно, то есть содержимое *ecx/cx* равно 0, то перейти к ячейке, обозначенной операндом метка;
- если проверяемое условие ложно, то есть содержимое *ecx/cx* не равно 0, то передать управление следующей за jcxz/jecxz команде программы.

Состояние флагов после выполнения команды:

11	07	06	05	04	03	02	01	00
OF	SF	ZF	0	AF	0	PF	1	CF
?	?	?		r		?		r

Применение (кроме jcxz/jecxz):
Команды условного перехода удобно применять для проверки различных условий, возникающих в ходе выполнения программы. Как известно, многие команды формируют признаки результатов своей работы в регистре eflags/flags. Это обстоятельство и используется командами условного перехода для работы. Ниже приведены перечень

команд условного перехода, анализируемые ими флаги и соответствующие им логические условия перехода.

Команда	Состояние проверяемых флагов	Условие перехода
JA	CF = 0 и ZF = 0	если выше
JAE	CF = 0	если выше или равно
JB	CF = 1	если ниже
JBE	CF = 1 или ZF = 1	если ниже или равно
JC	CF = 1	если перенос
JE	ZF = 1	если равно
JZ	ZF = 1	если 0
JG	ZF = 0 и SF = OF	если больше
JGE	SF = OF	если больше или равно
JL	SF <> OF	если меньше
JLE	ZF=1 или SF <> OF	если меньше или равно
JNA	CF = 1 и ZF = 1	если не выше
JNAE	CF = 1	если не выше или равно
JNB	CF = 0	если не ниже
JNBE	CF=0 и ZF=0	если не ниже или равно
JNC	CF = 0	если нет переноса
JNE	ZF = 0	если не равно
JNG	ZF = 1 или SF <> OF	если не больше
JNGE	SF <> OF	если не больше или равно
JNL	SF = OF	если не меньше
JNLE	ZF=0 и SF=OF	если не меньше или равно
JNO	OF=0	если нет переполнения
JNP	PF = 0	если количество единичных битов результата нечетно (нечетный паритет)
JNS	SF = 0	если знак плюс (знаковый (старший) бит результата равен 0)
JNZ	ZF = 0	если нет нуля
JO	OF = 1	если переполнение
JP	PF = 1	если количество единичных битов результата четно (четный паритет)
JPE	PF = 1	то же, что и JP, то есть четный паритет
JPO	PF = 0	то же, что и JNP
JS	SF = 1	если знак минус (знаковый (старший) бит результата равен 1)
JZ	ZF = 1	если ноль

Логические условия "больше" и "меньше" относятся к сравнениям целочисленных значений со знаком, а "выше и "ниже" — к сравнениям целочисленных значений без знака. Если внимательно посмотреть, то у многих команд можно заметить одинаковые

значения флагов для перехода. Это объясняется наличием нескольких ситуаций, которые могут вызвать одинаковое состояние флагов. В этом случае с целью удобства ассемблер допускает несколько различных мнемонических обозначений одной и той же машинной команды условного перехода. Эти команды ассемблера по действию абсолютно равнозначны, так как это одна и та же машинная команда. Изначально в микропроцессоре i8086 команды условного перехода могли осуществлять только короткие переходы в пределах -128...+127 байт, считая от следующей команды. Начиная с микропроцессора i386, эти команды уже могли выполнять любые переходы в пределах текущего сегмента команд. Это стало возможным за счет введения в систему команд микропроцессора дополнительных машинных команд. Для реализации межсегментных переходов необходимо комбинировать команды условного перехода и команду безусловного перехода jmp. При этом можно воспользоваться тем, что практически все команды условного перехода парные, то есть имеют команды, проверяющие обратные условия.
Применение jcxz/jecxz:

Команда	Состояние флагов в eflags/flags	Условие перехода
JCXZ	не влияет	если регистр CX=0
JECXZ	не влияет	если регистр ECX=0

Команду jcxz/jecxz удобно использовать со всеми командами, использующими регистр *ecx/cx* для своей работы. Это команды организации цикла и цепочечные команды. Очень важно отметить то, что команда jcxz/jecxz, в отличие от других команд перехода, может выполнять только близкие переходы в пределах -128...+127 байт, считая от следующей команды. Поэтому для нее особенно актуальна проблема передачи управления далее чем в указанном диапазоне. Для этого можно привлечь команду безусловного перехода jmp. Например, команду jcxz/jecxz можно использовать для предварительной проверки счетчика цикла в регистре cx для обхода цикла, если его счетчик нулевой.

```
...
        jcxz    m1      ;обойти цикл,
если cx=0
cycl:
;некоторый цикл
        loop    cycl
m1:     ...
```

См. также: уроки 10, 11 и команду *jmp*

JMP

(JuMP)
Переход безусловный

Схема команды: jmp метка
Назначение: используется в программе для организации безусловного перехода как внутри текущего сегмента команд, так и за его пределы. При определенных условиях в защищенном режиме работы команда jmp может использоваться для переключения задач.

Синтаксис
Алгоритм работы:
Команда jmp в зависимости от типа своего операнда изменяет содержимое либо только одного регистра eip, либо обоих регистров cs и eip:

- если операнд в команде jmp — метка в текущем сегменте команд (a8, 16, 32), то ассемблер формирует машинную команду, операнд которой является значением со знаком, являющимся смещением перехода относительно следующей за jmp команды. При этом виде перехода изменяется только регистр eip/ip;
- если операнд в команде jmp — символический идентификатор ячейки памяти (m16, 32, 48), то ассемблер предполагает, что в ней находится адрес, по которому необходимо передать управление. Этот адрес может быть трех видов:
    - значением абсолютного смещения метки перехода относительно начала сегмента кода. Размер этого смещения может быть 16 или 32 бит в зависимости от режима адресации;
    - дальним указателем на метку перехода в реальном и защищенном режимах, содержащим два компонента адреса — сегментный и смещение. Размеры этих компонентов также зависят от установленного режима адресации (use16 или use32). Если текущим режимом является use16, то адрес сегмента и смещение занимают по 16 бит, причем смещение располагается в младшем слове двойного слова, отводимого под этот полный адрес метки перехода. Если текущим режимом является use32, то адрес сегмента и смещение занимают, соответственно, 16 и 32 бит, — в младшем двойном слове находится смещение, в старшем — адрес сегмента;
    - адресом в одном из 16 или 32-разрядных регистров — этот адрес представляет собой абсолютное смещение метки, на которую необходимо передать управление, относительно начала сегмента команд.

Для понимания различий механизмов перехода в реальном и защищенном режимах нужно помнить следующее. В реальном режиме микропроцессор просто изменяет cs и eip/ip в соответствии с содержимым указателя в памяти. В защищенном режиме микропроцессор предварительно анализирует байт прав доступа AR в дескрипторе, номер которого определяется по содержимому сегментной части указателя. В зависимости от состояния байта AR микропроцессор выполняет либо переход, либо переключение задач.
Состояние флагов после выполнения команды (за исключением случая переключения задач):

выполнение команды не влияет на флаги

Применение:
Команду jmp применяют для осуществления ближних и дальних безусловных переходов без сохранения контекста точки перехода.
См. также: урок 10, команды call, jcc

LAHF

(Load AH register from register Flags)
Загрузка регистра AH флагами из регистра eFlags/Flags

Схема команды: lahf

Назначение: извлечение содержимого младшего байта регистра eflags/flags, в котором содержатся пять флагов: cf, pf, af, zf и sf.

Синтаксис
Алгоритм работы:
команда загружает регистр ah содержимым младшего байта регистра eflags/flags.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Из-за того, что регистр флагов непосредственно недоступен, команду lahf можно применять для анализа и последующего изменения командой sahf состояния некоторых флагов регистра eflags/flags.

```
;сбросить в ноль флаг cf
        lahf
        and     ah,11111110b
        sahf
```

См. также: команду sahf

LDS/LES/LFS/LGS/LSS

(Load pointer into ds/es/fs/gs/ss segment register)
Загрузка сегментного регистра ds/es/fs/gs/ss указателем из памяти

Схема команды:

lds приемник,источник
les приемник,источник
lfs приемник,источник
lgs приемник,источник
lss приемник,источник

Назначение: получение полного указателя в виде сегментной составляющей и смещения.

Синтаксис
Алгоритм работы:
Алгоритм работы команды зависит от действующего режима адресации (use16 или use32):

- если use16, то загрузить первые два байта из ячейки памяти источник в 16-разрядный регистр, указанный операндом приемник. Следующие два байта в области источник должны содержать сегментную составляющую некоторого адреса; они загружаются в регистр ds/es/fs/gs/ss;
- если use32, то загрузить первые четыре байта из ячейки памяти источник в 32-разрядный регистр, указанный операндом приемник. Следующие два байта в области источник должны содержать сегментную составляющую, или селектор, некоторого адреса; они загружаются в регистр ds/es/fs/gs/ss.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Таким образом, с помощью данных команд в паре регистров ds/es/fs/gs/ss и приемник оказывается полный адрес некоторой ячейки памяти. Это обстоятельство можно использовать, к примеру, при работе с цепочечными командами, где существуют жесткие соглашения на размещение адресов обрабатываемых строк. Помните, что любая загрузка сегментного регистра приводит к обновлению соответствующего теневого регистра (см. урок 16). Смотрите также описание команды cmps с примером использования.
См. также: уроки 5, 7, 11, команды lea и операторы ассемблера seg и offset

LEA

(Load Effective Address)
Загрузка эффективного адреса

Схема команды: lea приемник,источник
Назначение: получение эффективного адреса (смещения) источника.

Синтаксис
Алгоритм работы:
алгоритм работы команды зависит от действующего режима адресации (use16 или use32):

- если use16, то в регистр приемник загружается 16-битное значение смещения операнда источник;
- если use32, то в регистр приемник загружается 32-битное значение смещения операнда источник.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Данная команда является альтернативой оператору ассемблера offset. В отличие от offset команда lea допускает индексацию операнда, что позволяет более гибко организовать адресацию операндов.

```
;загрузить в регистр bx адрес пятого
элемента массива mas
.data
mas     db      10 dup (0)
.code
...
        mov     di,4
        lea     bx,mas[di]
;или
        lea     bx,mas[4]
;или
        lea     bx,mas+4
```

См. также: уроки 5, 7, 11 и команды lea, lds, les, lss, lgs, lfs, операторы ассемблера seg и offset

LEAVE

(LEAVE from procedure)
Выход из процедуры

Схема команды: leave
Назначение: удаление из стека области локальных (динамических) переменных, выделенной командой enter.

Синтаксис
Алгоритм работы:
команда выполняет обратные команде enter действия:

- содержимое ebp/bp копируется в esp/sp, тем самым восстанавливается значение esp/sp, которое было до вызова данной процедуры. С другой стороны, восстановление старого значения esp/sp означает освобождение пространства в стеке, отведенного для завершающейся процедуры (локальные переменные процедуры уничтожаются);
- из стека восстанавливается содержимое ebp/bp, которое было до входа в процедуру. После этого действия значение esp/sp также становится таким, каким оно было до входа в процедуру.

В результате этих двух действий также восстанавливается кадр стека, если он был, вызывающей программы.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда leave не имеет операндов и выполняет обратные команде enter действия. Эта команда должна находиться непосредственно перед командой ret, которая в зависимости от соглашений конкретного языка по вызову процедур удаляет или не удаляет аргументы из стека (см. урок 14).

```
.286
proc1   proc
        enter   16,0
...
        leave
        ret
proc1   endp
```

См. также: урок 14 и команды enter, ret/retf

LGDT

(Load Global Descriptor Table)
Загрузка регистра глобальной дескрипторной таблицы

Схема команды: lgdt источник
Назначение: загрузка регистра gdtr значениями базового адреса и размера глобальной дескрипторной таблицы GDT.

Синтаксис
Алгоритм работы:
команда выполняет загрузку 16 бит размера и 32 бит значения базового адреса начала таблицы GDT в памяти в системный регистр gdtr. Эта загрузка производится в соответствии с форматом этого регистра (см. урок 16). *Состояние флагов после выполнения команды:*

выполнение команды не влияет на флаги

Применение:
Команду lgdt применяют при подготовке к переходу в защищенный режим для загрузки системного регистра gdtr. В качестве операнда в команде указывается адрес области в формате 16+32. Младшее слово области — размер GDT, двойное слово по старшему адресу — значение базового адреса начала этой таблицы. Данные два компонента должны быть сформированы в памяти заранее.

```
.286
;структура для описания
псевдодескриптора gdtr
point   STRUC
lim     dw      0
adr     dd      0
 ENDS
.data
point_gdt       point
.code
...
;загружаем gdtr
        xor     eax,eax
        mov     ax,gdt_seg
        shl     eax,4
        mov     point_gdt.adr,eax
        lgdt    point_gdt
...
```

См. также: уроки 16, 17 и команду sgdt

LIDT

(Load Interrupt Descriptor Table)
Загрузка регистра глобальной дескрипторной таблицы

Схема команды: lidt источник

Назначение: загрузка регистра idtr значениями базового адреса и размера глобальной дескрипторной таблицы IDT.

Синтаксис
Алгоритм работы:
Команда lidt аналогична lgdt, но для дескрипторной таблицы прерываний IDT (см. урок 17).
Состояние флагов после выполнения команды:

```
выполнение команды не влияет на флаги
```

Применение:
Команду lidt применяют при подготовке к переходу в защищенный режим для загрузки системного регистра idtr. В качестве операнда в команде указывается адрес области в формате 16+32. Младшее слово области — размер IDT, двойное слово по старшему адресу — значение базового адреса начала этой таблицы. Два данных компонента должны быть сформированы в памяти заранее.

```
.386
;структура для описания
псевдодескрипторов gdtr и idtr
```

```
point   STRUC
lim     dw      0
adr     dd      0
 ENDS
.data
point_idt       point
.code
...
;загружаем idtr
        xor     eax,eax
        mov     ax,IDT_SEG
        shl     eax,4
        mov     point_idt.adr,eax
        lidt    point_idt
...
```

См. также: урок 17 и команду sidt

LODS/LODSB/LODSW/LODSD

(LOad String Byte/Word/Double word operands)
Загрузка строки байтов/слов/двойных слов

Схема команды:

lods источник
lodsb
lodsw
lodsd

Назначение: загрузка элемента из последовательности (цепочки) в регистр-аккумулятор al/ax/eax.

Синтаксис
Алгоритм работы:

- загрузить элемент из ячейки памяти, адресуемой парой ds:esi/si, в регистр al/ax/eax. Размер элемента определяется неявно (для команды lods) или явно в соответствии с применяемой командой (для команд lodsb, lodsw, lodsd);
- изменить значение регистра si на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:
  - df=0 — значение положительное, то есть просмотр от начала цепочки к ее концу;
  - df=1 — значение отрицательное, то есть просмотр от конца цепочки к ее началу.

Состояние флагов после выполнения команды:

```
выполнение команды не влияет на флаги
```

Применение:
Команды извлекают элемент из ячейки памяти в один из регистров. Перед командой lods можно указать префикс повторения rep, но в этом нет особого смысла, так как обычно эту команду используют в некотором цикле для просмотра некоторой цепочки с элементами фиксированного размера.

```
str     db      ...
```

```
...
        cld
        lea     si,str
        lodsb   ;загрузить первый байт
из str в al
```

См. также: урок 11 и команды ins/insb/insw/insd, cmps/cmpsb/cmpsw/cmpsd, movs/movsb/movsw/movsd, outs, scas/scasb/scasw/scasd, stos/stosb/stosw/stosd, rep/repe/repz/repne/repnz

LOOP

(LOOP control by register cx)
Управление циклом по cx

Схема команды: loop метка
Назначение: организация цикла со счетчиком в регистре cx.

Синтаксис
Алгоритм работы:

- выполнить декремент содержимого регистра *ecx/cx*;
- анализ регистра *ecx/cx*:
  - если *ecx/cx*=0, передать управление следующей за loop команде;
  - если *ecx/cx*=1, передать управление команде, метка которой указана в качестве операнда loop.

Состояние флагов после выполнения команды:

```
выполнение команды не влияет на флаги
```

Применение:
Команду loop применяют для организации цикла со счетчиком. Количество повторений цикла задается значением в регистре *ecx/cx* перед входом в последовательность команд, составляющих тело цикла. Помните о двух важных моментах:

- для предотвращения выполнения цикла при нулевом *ecx/cx* используйте команду jecxz/jcxz. Если этого не сделать, то при изначально нулевом *ecx/cx* цикл повторится 4 294 967 295/65 536 раз;
- смещение метки, являющейся операндом loop, не должно выходить из диапазона -128...+127 байт. Это смещение, как и в командах условного перехода, является относительным от значения счетчика адреса следующей за loop команды.

```
        mov     cx,10
...
        jcxz    m1
cycl:
;тело цикла
        loop    cycl
m1:
```

См. также: урок 10 и команды jecxz/jcxz, loope/loopz, loopne/loopnz

LOOPE/LOOPZ
LOOPNE/LOOPNZ

(LOOP control by register cx not equal 0 and ZF=1)
(LOOP control by register cx not equal 0 and ZF=0)
Управление циклом по cx с учетом значения флага ZF

Схема команды:

loope/loopz метка
loopne/loopnz метка

Назначение: организация цикла со счетчиком в регистре cx с учетом флага zf.

Синтаксис
Алгоритм работы:

- выполнить декремент содержимого регистра *ecx/cx*;
- проанализировать регистр *ecx/cx*:
  - если *ecx/cx*=0, передать управление следующей за loopxx команде;
  - если *ecx/cx*=1, передать управление команде, метка которой указана в качестве операнда loopxx;
- анализ флага zf:
  - если zf=0, для команд loope/loopz это означает выход из цикла, для команд loopne/loopnz — переход к началу цикла;
  - если zf=1, для команд loope/loopz это означает переход к началу цикла, для команд loopne/loopnz — выход из цикла.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команды loopxx удобно использовать вместе с командами, которые в результате своей работы меняют значение флага zf. Типичный пример — команда сравнения cmp.

```
;найти первый пробел в строке символов
str     db      'Найти первый пробел'
str_size=$-str
...
        cld
        mov     cx,str_size
        lea     si,str
cycl:
        lodsb
        cmp     al,' '
        loopne  cycl
        jcxz    m1      ;переход, если
пробелов нет
        dec     si      ;в si – адрес
пробела в строке str
...
m1
```

См. также: уроки 8, 10, 11 и команду loop

MOV

(MOVe operand)
Пересылка операнда

Схема команды: mov приемник,источник
Назначение: пересылка данных между регистрами или регистрами и памятью.

Синтаксис
Алгоритм работы:
копирование второго операнда в первый операнд.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда mov применяется для различного рода пересылок данных, при этом, несмотря на всю простоту этого действия, необходимо помнить о некоторых ограничениях и особенностях выполнения данной операции:

- направление пересылки в команде mov всегда справа налево, то есть из второго операнда в первый;
- значение второго операнда не изменяется;
- оба операнда не могут быть из памяти (при необходимости можно использовать цепочечную команду movs);
- лишь один из операндов может быть сегментным регистром;
- желательно использовать в качестве одного из операндов регистр al/ax/eax, так как в этом случае TASM генерирует более быструю форму команды mov.

```
        mov     al,5
        mov     bl,al
        mov     bx,ds
```

См. также: урок 10 и команды movs, lods/lodsb/lodsw/lodsd, stos/stosb, stosw/stosd

MOV

(MOVe operand to/from system registers)
Пересылка операнда в системные регистры (или из них)

Схема команды: mov приемник,источник
Назначение: пересылка данных между регистрами или регистрами и памятью.

Синтаксис
Алгоритм работы:
копирование второго операнда в первый.
Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команда mov применяется для обмена данными между системными регистрами. Это одна из немногих возможностей доступа к содержимому этих регистров. Данную команду можно использовать только на нулевом уровне привилегий либо в реальном режиме работы микропроцессора.

```
.286
;переключение микропроцессора в
защищенный
режим36:
        mov     eax,cr0
        bts     eax,0
        mov     cr0,eax
```

См. также: уроки 16, 17 и команды mov, bts

MOVS/MOVSB/MOVSW/MOVSD

(MOVe String Byte/Word/Double word)
Пересылка строк байтов/слов/двойных слов

Схема команды:

movs приемник,источник
movsb
movsw
movsd

Назначение: пересылка элементов двух последовательностей (цепочек) в памяти.

Синтаксис
Алгоритм работы:

- выполнить копирование байта, слова или двойного слова из операнда источника в операнд приемник, при этом адреса элементов предварительно должны быть загружены:
  - адрес источника — в пару регистров ds:esi/si (*ds* по умолчанию, допускается замена сегмента);
  - адрес приемника — в пару регистров es:edi/di (замена сегмента не допускается);
- в зависимости от состояния флага df изменить значение регистров esi/si и edi/di:
  - если df=0, то увеличить содержимое этих регистров на длину структурного элемента последовательности;
  - если df=1, то уменьшить содержимое этих регистров на длину структурного элемента последовательности;
- если есть префикс повторения, то выполнить определяемые им действия (см. команду rep).

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команды пересылают элемент из одной ячейки памяти в другую. Размеры пересылаемых элементов зависят от применяемой команды. Команда movs может работать с элементами размером в байт, слово, двойное слово. В качестве операндов в команде указываются идентификаторы последовательностей этих элементов в памяти. Реально эти идентификаторы используются лишь для получения типов элементов последовательностей, а их адреса должны быть предварительно загружены в указанные выше пары регистров. Транслятор, обработав команду movs и выяснив тип операндов, генерирует одну из машинных команд movsb, movsw или movsd. Машинного аналога для команды movs нет. Для адресации операнда приемник обязательно должен использоваться регистр es.
Для того чтобы эти команды можно было использовать для пересылки последовательности элементов, имеющих размерность байт, слово, двойное слово, необходимо использовать префикс rep. Префикс rep заставляет циклически выполняться команды пересылки до тех пор, пока содержимое регистра *ecx/cx* не станет равным нулю.

```
str1    db      'str1 копируется в
str2'
len_str1=$-str1
a_str1  dd      str1
str2    db      len_str1 dup (' ')
a_str2  dd      str2
...
        mov     cx,len_str1
        lds     si,str1
        les     di,str2
        cld
rep     movsb
```

См. также: урок 11 и команды cmps/cmpsb/cmpsw/cmpsd, ins/insb/insw/insd, lods/lodsb/lodsw/lodsd, outs, scas/scasb/scasw/scasd, stos/stosb/stosw/stosd, rep/repe/repz/repne/repnz

MOVSX

(MOVe and Sign eXtension)
Пересылка со знаковым расширением

Схема команды: movsx приемник,источник
Назначение: преобразование элементов со знаком меньшей размерности в эквивалентные им элементы со знаком большей размерности.

Синтаксис
Алгоритм работы:

- считать содержимое источника;
- записать содержимое операнда источника в операнд приемник, начиная с младших разрядов источника;
- распространить значение знакового разряда источника на свободные старшие разряды операнда назначения.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команду movsx обычно используют для получения эквивалентного, но большего по размеру операнда со знаком. Это может понадобиться для приведения размера операнда к нужному значению с целью обеспечения работы следующих команд программы:

```
        mov     al,0ffh
        movsx   bx,al   ;bx=0ffffh
```

См. также: урок 8 и команды *mov*, *movs*, *movzx*, *cbw*, *cwd*, *cdq*

MOVZX

(MOVe and Zero eXtension)
Пересылка с нулевым расширением

Схема команды: movzx приемник,источник
Назначение: преобразование элементов без знака меньшей размерности в эквивалентные им элементы без знака большей размерности.

Синтаксис
Алгоритм работы:

- считать содержимое источника;
- записать содержимое операнда источника в операнд приемник, начиная с его младших разрядов;
- распространить двоичный нуль на свободные старшие разряды операнда назначения.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команду movzx обычно используют для получения эквивалентного, но большего по размеру операнда без учета знака. Она может быть использована для согласования операндов различной размерности. Но не следует думать, что все эти разнотипные пересылки делает одна машинная команда. На самом деле существует несколько машинных команд, каждая из которых работает со своими размерами операндов. Генерацию же нужной команды обеспечивает транслятор на основе анализа исходного текста программы.

```
.data
sl      db      ?
.code
...
        mov     al,0ffh
        movzx   bx,al   ;bx=00ffh
...
;или из памяти:
        movzx   eax,byte ptr sl
```

См. также: урок 8 и команды mov, movs/movsb/ movsw/movsd, movsx, cbw, cwd, cdq

MUL

(MULtiply)
Умножение целочисленное без учета знака

Схема команды: mul множитель_1
Назначение: операция умножения двух целых чисел без учета знака.

Синтаксис
Алгоритм работы:
Команда выполняет умножение двух операндов без учета знаков. Алгоритм зависит от формата операнда команды и требует явного указания местоположения только одного сомножителя, который может быть расположен в памяти или в регистре. Местоположение второго сомножителя фиксировано и зависит от размера первого сомножителя:

- если операнд, указанный в команде — байт, то второй сомножитель должен располагаться в al;
- если операнд, указанный в команде — слово, то второй сомножитель должен располагаться в ax;
- если операнд, указанный в команде — двойное слово, то второй сомножитель должен располагаться в eax.

Результат умножения помещается также в фиксированное место, определяемое размером сомножителей:

- при умножении байтов результат помещается в ax;
- при умножении слов результат помещается в пару dx:ax;
- при умножении двойных слов результат помещается в пару edx:eax.

Состояние флагов после выполнения команды (если старшая половина результата нулевая):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
0	?	?	?	?	0

Состояние флагов после выполнения команды (если старшая половина результата ненулевая):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
1	?	?	?	?	1

Применение:
Команда mul выполняет целочисленное умножение операндов без учета их знаковых разрядов. Для этой операции необходимо наличие двух операндов-сомножителей, размещение одного из которых фиксировано, а другого задается операндом в команде. Контролировать размер результата удобно используя флаги cf и of.

```
mn_1    db      15
mn_2    db      25
...
        mov     al,mn_1
        mul     mn_2
```

См. также: урок 8 и команду *imul*

NEG

(NEGate operand)
Изменить знак операнда

Схема команды: neg источник
Назначение: изменение знака (получение двоичного дополнения) источника.

Синтаксис
Алгоритм работы:

- выполнить вычитание (0 – источник) и поместить результат на место источника;
- если источник=0, то его значение не меняется.

Состояние флагов после выполнения команды (если результат нулевой):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	0

Состояние флагов после выполнения команды (если результат ненулевой):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	1

Применение:
Команда используется для формирования двоичного дополнения операнда в памяти или регистре. Операция двоичного дополнения предполагает инвертирование всех разрядов операнда с последующим сложением операнда с двоичной единицей. Если операнд отрицательный, то операция neg над ним означает получение его модуля.

```
        mov     al,2
        neg     al      ;al=0feh –
число -2 в дополнительном коде
```

См. также: уроки 6, 8 и команду not

NOP

(No OPeration)
Нет операции

Схема команды: nop

Назначение: пустая команда.

Синтаксис

Алгоритм работы:

не производит никаких действий.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:

Команда nop, занимая один байт, может использоваться для резервирования места в сегменте кода или организации программной задержки. В качестве иллюстрации можно обратиться к примеру, приведенному в описании команды hlt. В этом примере команду nop можно использовать вместо jmp $+2. Назначение jmp $+2 в этом фрагменте — задержка для синхронизации работы микропроцессора и аппаратуры компьютера.

NOT

(NOT operand)
Инвертирование операнда

Схема команды: not источник

Назначение: инвертирование всех битов операнда источник.

Синтаксис

Алгоритм работы:

инвертировать все биты операнда источника: из 1 в 0, из 0 в 1.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:

Команду not можно использовать для изменения байта, выполняющего роль некоторого флага, с целью отслеживания некоторых логических условий в программе. Но такой способ не оптимален, эту ситуацию мы обсуждали в книге на уроках 9 и 12.

```
flag    db      0ffh ;значение флага –
истина
...
cycl:
...
        cmp     flag,0
        je      m1
...
m1:     not     flag    ;установить
флаг в истину
```

См. также: уроки 9, 12 и команду neg

OR

(logical OR)
Логическое включающее ИЛИ

Схема команды: or приемник,маска
Назначение: операция логического ИЛИ над битами операнда назначения.

Синтаксис
Алгоритм работы:

- выполнить операцию логического ИЛИ над битами операнда назначения, используя в качестве маски второй операнд — маска. При этом бит результата равен 0, если соответствующие биты операндов маска и назначения равны 0, в противном случае бит равен 1;
- записать результат операции в источник (операнд маска остается неизменным);
- установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
0	r	r	?	r	0

Применение:
Команду or можно использовать для работы с операндами на уровне битов. Типичное использование команды — установка определенных разрядов первого операнда в единицу.

```
        mov     al,01h
        or      bl,al   ;установить
нулевой бит в 1
```

См. также: урок 9 и команды and, xor, not

OUT

(OUT operand to port)
Вывод операнда в порт

Схема команды: out ном_порта,аккумулятор
Назначение: вывод значения в порт ввода-вывода.

Синтаксис
Алгоритм работы:
Передать байт, слово, двойное слово из регистра al/ax/eax в порт, номер которого определяется первым операндом.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда применяется для прямого управления оборудованием компьютера посредством

портов. Номер порта задается первым операндом в виде непосредственного значения или значения в регистре dx. Непосредственным значением можно задать порт с номером в диапазоне 0...255. Для указания порта с большим номером используется регистр dx. Размер данных определяется размерностью второго операнда и может быть байтом, словом или двойным словом.

```
        out     64h,al
```

См. также: уроки 2, 7, 16, 17 и команды in, ins/insb/insw/insd, outs

OUTS/OUTSB/OUTSW/OUTSD

(OUTput Byte/Word/Double word String to port)
Вывод строки байтов/слов/двойных слов в порт

Схема команды:

outs порт,источник
outsb
outsw
outsd

Назначение: вывод в порт из памяти последовательности байт, слов, двойных слов.

Синтаксис
Алгоритм работы:

- передать данные в порт ввода-вывода, номер которого загружен в регистр dx, из ячейки памяти по адресу ds:esi/si;
- в зависимости от состояния флага df изменить значение регистров esi/si:
  - если df=0, то увеличить содержимое этих регистров на длину структурного элемента последовательности;
  - если df=1, то уменьшить содержимое этих регистров на длину структурного элемента последовательности;
- при наличии префикса выполнить определяемые им deiqrbh (см. команду rep/repe/repz/repne/repnz).

Состояние флагов после выполнения команды:

```
выполнение команды не влияет на флаги
```

Применение:
Команда выводит данные в порт ввода-вывода, номер которого загружен в регистр dx, из ячейки памяти по адресу ds:esi/si (допускается замена сегмента). Недопустимо задание номера порта в команде в виде непосредственного операнда — для этого используется регистр dx. Размеры вводимых элементов зависят от применяемой команды. Команда outs может работать с элементами размером в байт, слово или двойное слово. В качестве операнда в команде указывается символическое имя ячейки памяти, из которой элемент выводится в порт ввода-вывода. Реально символическое имя используется лишь для получения типа элемента последовательности, а ее адрес должен быть предварительно загружен в пару регистров ds:esi/si. Транслятор, обработав команду outs и выяснив тип операндов, генерирует одну из машинных команд outsb, outsw или outsd. Машинного аналога для команды outs нет.
Для того чтобы эти команды можно было использовать для вывода в порт последовательности элементов, имеющих размерность байт, слово или двойное слово,

необходимо использовать префикс rep. Он заставляет циклически выполняться команду вывода в порт до тех пор, пока содержимое регистра *ecx/cx* не станет равным нулю.

```
.286
;вывести последовательность 10 байт в
порт 300h
;(номер порта взят условно)
str_10  db      10 dup(0)
adr_str dd      str_10
        lds     si,adr_str
        mov     dx,300h
rep     outsb
```

См. также: уроки 2, 7, 11 и команды cmps/cmpsb/cmpsw/cmpsd, lods/lodsb/lodsw/lodsd, movs/movsb/movsw/movsd, ins/insb/insw/insd, scas/scasb/scasw/scasd, stos/stosb/stosw/stosd, rep/repe/repz/repne/repnz

POP

(POP operand from the stack)
Извлечение операнда из стека

Схема команды: pop приемник
Назначение: извлечение слова или двойного слова из стека.

Синтаксис
Алгоритм работы:
Алгоритм работы команды зависит от установленного атрибута размера адреса — use16 или use32:

- загрузить в приемник содержимое вершины стека (адресуется парой ss:esp/sp);
- увеличить содержимое esp/sp на 4 (2 байта) для use32 (соответственно для use16).

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда применяется для восстановления содержимого вершины стека в регистр, ячейку памяти или сегментный регистр. Заметим, что недопустимо восстановление значения в сегментный регистр cs.

```
my_proc proc    near
        push    ax
        push    bx
;тело процедуры, в которой изменяется
содержимое
;регистров ax и bx
...
        pop     bx
        pop     ax
        ret
        endp
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды popa, popad, popf, popfd, push, pusha, pushad, pushf, pushfd

POPA

(POP All general registers from the stack)
Извлечение всех регистров общего назначения из стека

Схема команды: popa
Назначение: извлечение из стека регистров общего назначения di, si, bp, sp, bx, dx, cx, ax.

Синтаксис
Алгоритм работы:

- извлечь из стека последовательно значения и загрузить ими регистры общего назначения di, si, bp, sp, bx, dx, cx, ax. Содержимое di восстанавливается первым. Содержимое sp извлекается, но не восстанавливается;
- увеличить значение указателя стека esp/sp на 16.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда popa по принципу работы является обратной команде pusha и используется для восстановления содержимого всех регистров общего назначения значениями из стека. Эту команду можно использовать в процедурах и программах обработки прерываний для восстановления регистров общего назначения прерванной программы.

```
.386
my_proc proc    near
        pusha
;тело процедуры, в которой изменяется
;содержимое регистров общего назначения
...
        popa
        ret
        endp
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popad, popf, popfd, push, pusha, pushad, pushf, pushfd

POPAD

(POP All general Double word registers from the stack)
Извлечение всех 32-разрядных регистров общего назначения из стека

Схема команды: popad
Назначение: извлечение из стека регистров общего назначения edi, esi, ebp, esp, ebx, edx, ecx, eax.

Синтаксис
Алгоритм работы:

- извлечь из стека последовательно значения и загрузить ими 32-разрядные регистры общего назначения edi, esi, ebp, esp, ebx, edx, ecx, eax. Содержимое edi восстанавливается первым. Содержимое esp извлекается но не восстанавливается;
- увеличить значение указателя стека esp на 32.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда popad по принципу работы является обратной команде pushad и используется для восстановления всех 32-разрядных регистров общего назначения. Эту команду можно использовать в процедурах и программах обработки прерываний для восстановления регистров общего назначения прерванной программы.

```
.386
my_proc proc    near
        pushad
;тело процедуры, в которой изменяется
;содержимое регистров общего назначения
...
        popad
        ret
        endp
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popa, popf, popfd, push, pusha, pushad, pushf, pushfd

POPF

(POP Flags register from the stack)
Извлечение регистра флагов из стека

Схема команды: popf
Назначение: извлечение из стека слова и восстановление его в регистр флагов flags.

Синтаксис
Алгоритм работы:

- извлечь из вершины стека слово и поместить его в регистр flags;
- увеличить значение указателя стека esp на 2.

Состояние флагов после выполнения команды:

14	1312	11	10	09	08	07	06	04	02	00
NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
r	r	r	r	r	r	r	r	r	r	r

Применение:
Команда popf по принципу работы является обратной команде pushf и используется для восстановления из стека содержимого регистра флагов eflags. Возможным вариантом использования этой команды являются программы обработки прерываний или другие случаи, в которых необходимо сохранять некоторый локальный контекст процесса

вычисления. Из-за того, что регистр eflags/flags непосредственно недоступен, команда popf является одной из немногих возможностей влияния на его содержимое.

```
;установить значение регистра flags в
03h
        mov     ax,3h
        push    ax
        popf
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popa, popad, popfd, push, pusha, pushad, pushf, pushfd

POPFD

(POP eFlags Double word register from the stack)
Извлечение расширенного регистра флагов из стека

Схема команды: popfd

Назначение: извлечение из стека двойного слова и восстановление его в регистр флагов eflags.

Синтаксис
Алгоритм работы:

- извлечь из вершины стека двойное слово и поместить его в регистр eflags;
- увеличить значение указателя стека esp на 4.

Состояние флагов после выполнения команды:

17	16	14	1312	11	10	09	08	07	06	04	02	00
VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
0	r	r	r	r	r	r	r	r	r	r	r	r

Применение:
Команда popfd по принципу работы является обратной командой команде pushfd и используется для восстановления из стека содержимого регистра флагов eflags. Необходимо отметить, что команда popfd не влияет на состояние флагов vm и rf.

```
.386
;установить значение регистра eflags в
03h
        mov     eax,3h
        push    eax
        popfd   eax     ;установить
новое значение eflags
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popa, popad, popf, push, pusha, pushad, pushf, pushfd

PUSH

(PUSH operand onto stack)
Размещение операнда в стеке

Схема команды: push источник
Назначение: размещение содержимого операнда источник в стеке.

Синтаксис
Алгоритм работы:

- уменьшить значение указателя стека esp/sp на 4/2 (в зависимости от значения атрибута размера адреса — use16 или use32);
- записать источник в вершину стека (адресуемую парой ss:esp/sp).

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда push используется совместно с командой pop для записи значений в стек и извлечения их из стека. Размер записываемых значений — слово или двойное слово. Также в стек можно записывать непосредственные значения. Заметьте, что в отличие от команды pop в стек можно включать значение сегментного регистра cs. Другой интересный момент связан с регистром sp. Команда push esp/sp записывает в стек значение esp/sp по состоянию до выдачи этой команды. В микропроцессоре i8086 по этой команде записывалось скорректированное значение sp. При записи в стек 8-битных значений для них все равно выделяется слово или двойное слово (в зависимости от use16 или use32).

```
my_proc proc    near
        push    ax
        push    bx
;тело процедуры, в которой изменяется
содержимое
;регистров ax и bx
...
        pop     bx
        pop     ax
        ret
        endp
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popa, popad, popf, popfd, pusha, pushad, pushf, pushfd

PUSHA

(PUSH All general registers onto stack)
Размещение всех регистров общего назначения в стеке

Схема команды: pusha
Назначение: размещение в стеке регистров общего назначения в следующей последовательности: ax, cx, dx, bx, sp, bp, si, di.

Синтаксис
Алгоритм работы:

- уменьшить значение указателя стека esp/sp на 32/16 (в зависимости от значения атрибута размера адреса — use16 или use32);
- включить в стек последовательно значения регистров общего назначения ax, cx, dx, bx, sp, bp, si, di.

Содержимое di при этом будет на вершине стека. В стек помещается содержимое sp по состоянию до выполнения команды.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда pusha используется совместно с командой popa для сохранения и восстановления всех регистров общего назначения. Эти команды удобно использовать при работе с процедурами, программами обработки прерываний, а также в других случаях для сохранения и восстановления регистров общего назначения как части контекста некоторого вычислительного процесса.

```
my_proc proc    near
        pusha
;тело процедуры, в которой изменяется
;содержимое регистров общего назначения
...
        popa
        ret
        endp
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popad, popf, popfd, push, popa, pushad, pushf, pushfd

PUSHAD

(PUSH All general Double word registers onto stack)
Размещение всех регистров общего назначения в стеке

Схема команды: pushad
Назначение: размещение в стеке регистров общего назначения в следующей последовательности: eax, ecx, edx, ebx, esp, ebp, esi, edi.

Синтаксис
Алгоритм работы:

- уменьшить значение указателя стека esp на 32;
- включить в стек последовательно значения регистров общего назначения eax, ecx, edx, ebx, esp, ebp, esi, edi. Содержимое edi при этом будет на вершине стека. Содержимое esp включается по состоянию на момент, предшествовавший выполнению данной команды.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команда pushad используется совместно с командой popad для сохранения и

восстановления всех регистров общего назначения. Эти команды используются аналогично командам popa и pusha.

```
.386
my_proc proc    near
        pushad
;тело процедуры, в которой изменяется
;содержимое регистров общего назначения
...
        popad
        ret
        endp
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popa, popf, popfd, push, pusha, popad, pushf, pushfd

PUSHF

(PUSH Flags register onto stack)
Размещение регистра флагов в стеке

Схема команды: pushf
Назначение: размещение в вершине стека (ss:sp) содержимого регистра флагов flags.

Синтаксис
Алгоритм работы:

- уменьшить значение указателя стека sp на 2;
- поместить в вершину стека содержимое регистра flags.

Состояние флагов после выполнения команды:

```
выполнение команды не влияет на флаги
```

Применение:
Команда pushf может использоваться для получения содержимого регистра флагов. Как известно, прямой доступ к регистру флагов невозможен, поэтому данная команда является одной из немногих команд, позволяющих получить доступ к регистру флагов как к содержимому обычного регистра. Обратное действие, то есть восстановление — возможно измененного слова — в регистр флагов, осуществляется командой popf. Эта команда может использоваться в программах обработки прерываний и в других случаях, когда необходимо сохранить локальный контекст процесса вычисления.

```
;извлечь значение регистра flags и
изменить
;значение флага cf на обратное
        pushf
        pop     ax
        xor     ax,01h
        push    ax
        popf
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popa, popad, popfd, push, pusha, pushad, popf, pushfd

PUSHFD

(PUSH eFlags Double word register onto stack)
Размещение расширенного регистра флагов в стеке

Схема команды: pushfd
Назначение: размещение в стеке содержимого регистра флагов eflags.

Синтаксис
Алгоритм работы:

- уменьшить значение указателя стека esp на 4;
- записать в вершину стека двойное слово, представляющее собой содержимое регистра eflags.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команды pushfd и popfd используются аналогично командам pushf и popf. Команда pushfd применяется для получения содержимого регистра флагов. Как известно, прямой доступ к регистру флагов невозможен, поэтому данная команда является одной из немногих команд, позволяющих получить доступ к регистру флагов как к содержимому обычного регистра. Обратное действие, то есть восстановление — возможно измененного слова — в регистр флагов, осуществляется командой popfd. Эта команда может использоваться в программах обработки прерываний или в других случаях, когда необходимо сохранить локальный контекст процесса вычисления.

```
.386
;извлечь значение регистра eflags и
изменить
;значение флага cf на обратное
        pushfd
        pop     eax
        xor     eax,01h
        push    eax
        popfd
```

См. также: уроки 7, 10, 14, 15, 16, 17 и команды pop, popa, popad, popf, popfd, push, pusha, pushad, pushf

RCL

(Rotate operand through Carry flag Left)
Циклический сдвиг операнда влево через флаг переноса

Схема команды: rcl операнд,количество_сдвигов
Назначение: операция циклического сдвига операнда влево через флаг переноса cf.

Синтаксис
Алгоритм работы:

- сдвиг всех битов операнда влево на один разряд, при этом старший бит операнда становится значением флага переноса cf;
- одновременно старое значение флага переноса cf вдвигается в операнд справа и становится значением младшего бита операнда;
- указанные выше два действия повторяются количество раз, равное значению второго операнда команды rcl.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Здесь обозначение ?r означает то, что анализ состояния флага имеет смысл при определенном сочетании операндов. В случае команды rcl флаг of представляет интерес, если сдвиг осуществляется на один разряд (см. ниже описание применения команды rcl).
Применение:
Команда rcl используется для циклического сдвига разрядов операнда влево. Особенность этого сдвига в том, что он происходит с некоторой задержкой, так как очередной сдвигаемый бит оказывается на некоторое время вне операнда. В это время можно произвести его извлечение и (или) подмену. Другой важный момент заключается в том, что для счетчика сдвига микропроцессор использует только пять младших разрядов операнда количество_разрядов. Таким образом, значение, большее 31, микропроцессором не допускается (аппаратно это ограничение реализуется тем, что игнорируются значения всех битов счетчика, кроме первых пяти). Обратите внимание на еще один интересный эффект, связанный с поведением флага of. В операциях сдвига на один разряд по изменению этого флага можно судить о факте изменения знакового (старшего) разряда операнда:

- of=1, если текущее значение флага cf и выдвигаемого бита операнда слева различны;
- of=0, если текущее значение флага cf и выдвигаемого бита операнда слева совпадают.

```
;сдвиг операнда, занимающего два
двойных слова
;на четыре разряда влево
ch_l    dd      ...     ;младшая часть
64-битного операнда
ch-2    dd      ...     ;старшая часть
64-битного операнда
...
        mov     cx,4    ;счетчик
сдвигов в cx
        mov     eax,ch_l
        mov     edx,ch_h
m1:     clc             ;очистка флага
cf
        rcl     eax,1   ;старший бит
eax в cf
        rcl     edx,1   ;cf в младший
бит edx, старший бит edx в cf
        loop    m1
```

См. также: урок 9 и команды rcr, rol, ror, sal, sar, shl, shr

RCR

(Rotate operand through Carry flag Right)
Циклический сдвиг операнда вправо через флаг переноса

Схема команды: rcr операнд,количество_сдвигов
Назначение: операция циклического сдвига операнда вправо через флаг переноса cf.

Синтаксис
Алгоритм работы:

- сдвиг всех битов операнда вправо на один разряд; при этом младший бит операнда становится значением флага переноса cf;
- одновременно старое значение флага переноса — в операнд слева и становится значением старшего бита операнда;
- указанные выше два действия повторяются количество раз, равное значению второго операнда команды rcr.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Здесь обозначение ?r означает то, что анализ состояния флага имеет смысл при определенном сочетании операндов. В случае команды rcr флаг of представляет интерес, если сдвиг осуществляется на один разряд (см. ниже описание применения команды rcr).
Применение:
Команда rcr используется для циклического сдвига разрядов операнда вправо. Особенность этого сдвига в том, что он происходит с некоторой задержкой, так как очередной сдвигаемый бит оказывается на некоторое время вне операнда. В это время можно произвести его извлечение и (или) подмену. Другой важный момент заключается в том, что для счетчика сдвига микропроцессор использует только пять младших разрядов операнда количество_разрядов. Таким образом, значение, большее 31, не допускается (аппаратно это ограничение реализуется тем, что игнорируются значения битов счетчика старше пятого). Обратите внимание на еще один интересный эффект, связанный с поведением флага of, — его значение имеет смысл только в операциях сдвига на один разряд и обусловлено тем, что по изменению этого флага можно судить о факте изменения знакового разряда операнда:

- of=1, если текущие (то есть до операции сдвига) значения флага cf и старшего, левого бита операнда различны;
- of=0, если текущие (то есть до операции сдвига) значения флага cf и старшего, левого бита операнда слева совпадают.

```
;подсчет числа единичных битов в
операнде
operand dw      ...
```

```
...
        mov     cx,16   ;размер
операнда
        xor     al,al   ;счетчик
единичных битов
cycl:   rcr     operand,1
        jc      $+4     ;переход, если
очередной выдвинутый бит равен 1
        jmp     $+4     ;переход, если
очередной выдвинутый бит равен 0
        inc     al      ;увеличить
счетчик единичных битов
        loop    cycl
```

См. также: урок 9 и команды rcl, rol, ror, sal, sar, shl, shr

REP/REPE/REPZ/REPNE/REPNZ

(REPeat string operation)
Повторить цепочечную операцию

Схема команды:

rep
repe
repz
repne
repnz

Назначение: указание условного и безусловного повторения следующей за данной командой цепочечной операции.

Синтаксис
Алгоритм работы:
Алгоритм работы зависит от конкретного префикса. Префиксы rep, repe и repz на самом деле имеют одинаковый код операции, их действия зависят от той цепочечной команды, которую они предваряют:

- rep используется перед следующими цепочечными командами и их краткими эквивалентами: movs, stos, ins, outs. Действия rep:
    1. анализ содержимого cx:
        - если cx<>0, то выполнить цепочечную команду, следующую за данным префиксом и перейти к шагу 2;
        - если cx=0, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по rep);
    2. уменьшить значение cx=cx–1 и вернуться к шагу 1;
- repe и repz используются перед следующими цепочечными командами и их краткими эквивалентами: cmps, scas. Действия repe и repz:
    1. анализ содержимого cx и флага zf:
        - если cx<>0 или zf<>0, то выполнить цепочечную команду, следующую за данным префиксом, и перейти к шагу 2;
        - если cx=0 или zf=0, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по rep);
    2. уменьшить значение cx=cx-1 и вернуться к шагу 1;

- repne и repnz также имеют один код операции и имеют смысл при использовании перед следующими цепочечными командами и их краткими эквивалентами: cmps, scas. Действия repne и repnz:
    1. анализ содержимого cx и флага zf:
        - если cx<>0 или zf=0, то выполнить цепочечную команду, следующую за данным префиксом и перейти к шагу 2;
        - если cx=0 или zf<>0, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по rep);
    2. уменьшить значение cx=cx–1 и вернуться к шагу 1.

Состояние флагов после выполнения команды:

Применение:
Команды rep, repe, repz, repne и repnz в силу специфики своей работы называются префиксами. Они имеют смысл только при использовании цепочечных операций, заставляя их циклически выполняться и тем самым без организации внешнего цикла обрабатывать последовательности элементов фиксированной длины. Большинство применяемых префиксов являются условными, то есть они прекращают работу цепочечной команды при выполнении определенных условий.
См. также: урок 11 и команды cmps/cmpsb/cmpsw/cmpsd, ins/insb/insw/insd, outs, movs/movsb/movsw/movsd, scas/scasb/scasw/scasd, stos/stosb /stosw/stosd

RET/RETF

(RETurn/RETurn Far from procedure)
Возврат ближний (дальний) из процедуры

Схема команды:

ret
ret число

Назначение: возврат управления из процедуры вызывающей программе.

Синтаксис
Алгоритм работы:
Работа команды зависит от типа процедуры:

- для процедур ближнего типа — восстановить из стека содержимое eip/ip;
- для процедур дальнего типа — последовательно восстановить из стека содержимое eip/ip и сегментного регистра cs.
- если команда ret имеет операнд, то увеличить содержимое esp/sp на величину операнда число; при этом учитывается атрибут режима адресации — use16 или use32:
    - если use16, то sp=(sp+число), то есть указатель стека сдвигается на число байт, равное значению число;
    - если use32, то sp=(sp+2*число), то есть указатель стека сдвигается на число слов, равное значению число.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команду ret необходимо применять для возврата управления вызывающей программе из процедуры, управление которой было передано по команде call. На самом деле микропроцессор имеет три варианта команды возврата ret - это ret, ее синоним retn, а также команда retf. Они отличаются типами процедур, в которых используются. Команды ret и retn служат для возврата из процедур ближнего типа. Команда retf — команда возврата для процедур дальнего типа. Какая конкретно команда будет использоваться, определяется компилятором; программисту лучше использовать команду ret и доверить транслятору самому сгенерировать ее ближний или дальний вариант. Количество команд ret в процедуре должно соответствовать количеству точек выхода из нее.
Некоторые языки высокого уровня, к примеру Pascal, требуют, чтобы вызываемая процедура очищала стек от переданных ей параметров. Для этого команда ret содержит необязательный параметр число, который, в зависимости от установленного атрибута размера адреса, означает количество байт или слов, удаляемых из стека по окончании работы процедуры.

```
my_proc proc
...
        ret     6
        endp
```

См. также: уроки 10, 14 и команду *call*

ROL

(Rotate operand Left)
Циклический сдвиг операнда влево

Схема команды: rol операнд,количество_сдвигов
Назначение: операция циклического сдвига операнда влево.

Синтаксис
Алгоритм работы:

- сдвиг всех битов операнда влево на один разряд, при этом старший бит операнда вдвигается в операнд справа и становится значением младшего бита операнда;
- одновременно выдвигаемый бит становится значением флага переноса cf;
- указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Применение:
Команда rol используется для циклического сдвига разрядов операнда влево. Отличие этого сдвига от rcl в том, что очередной сдвигаемый бит одновременно вдвигается в

операнд справа и становится значением флага cf. Так же, как и для других сдвигов, значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда количество_разрядов. Аналогично другим командам сдвига сохраняется эффект, связанный с поведением флага of, значение которого имеет смысл только в операциях сдвига на один разряд:

- если of=1, то текущее значение флага cf и выдвигаемого слева бита операнда различны;
- если of=0, то текущее значение флага cf и выдвигаемого слева бита операнда совпадают.

Этот эффект, как вы помните, обусловлен тем, что флаг of устанавливается в единицу всякий раз при изменении знакового разряда операнда.

```
;поменять местами половинки регистра
eax:
        mov     ax,0ffff0000h
        mov     cl,16
        rol     eax,cl  ;eax=0000ffffh
```

См. также: урок 9 и команды rcr, rcl, ror, sal, sar, shl, shr

ROR

Циклический сдвиг операнда вправо
ASCII-коррекция после сложения

Схема команды: ror операнд,количество_сдвигов
Назначение: операция циклического сдвига операнда вправо.

Синтаксис
Алгоритм работы:

- сдвиг всех битов операнда вправо на один разряд, при этом младший бит операнда вдвигается в операнд слева и становится значением старшего бита операнда;
- одновременно этот младший бит операнда становится значением флага переноса cf;
- старое значение флага переноса cf вдвигается в операнд слева и становится значением старшего бита операнда;
- указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Применение:
Команда ror используется для циклического сдвига разрядов операнда вправо. Отличие этого сдвига от rcr в том, что очередной сдвигаемый бит одновременно вдвигается в

операнд слева и становится значением флага cf. Так же, как и для других сдвигов, значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда количество_разрядов. Аналогично другим командам сдвига сохраняется эффект, связанный с поведением флага of, значение которого имеет смысл только в операциях сдвига на один разряд:

- если of=1, то текущее значение флага cf и вдвигаемого слева бита операнда различны;
- если of=0, то текущее значение флага cf и вдвигаемого слева бита операнда совпадают;

Этот эффект, как вы помните, обусловлен тем, что флаг of устанавливается в единицу всякий раз при изменении знакового разряда операнда.

```
;поместить четыре младших бита ax на
место старших битов:
        ror     ax,4
```

См. также: уроки 9 и команды rcl, rcr, ror, sal, sar, shl, shr

SAHF

(Store AH register into register Flags)
Загрузка регистра флагов eFlags/Flags из регистра AH

Схема команды: sahf

Назначение: запись содержимого регистра ah в младший байт регистра eflags/flags, в котором содержатся пять флагов cf, pf, af, zf и sf.

Синтаксис
Алгоритм работы:
Команда загружает младший байт регистра eflags/flags содержимым регистра ah. В битах 7, 6, 4, 2 и 0 регистра ah должны, соответственно, содержаться новые значения флагов sf, zf, af, pf и cf.
Состояние флагов после выполнения команды:

07	06	04	02	00
SF	ZF	AF	PF	CF
r	r	r	r	r

Применение:
Эта команда используется совместно с командой lahf. Из-за того, что регистр флагов непосредственно недоступен, сочетание этих команд можно применять для анализа — и, возможно, изменения — состояния некоторых флагов в регистре eflags/flags. Содержимое старшей части регистра флагов не изменяется.

```
;сбросить в ноль флаг cf
        lahf
        and     ah,11111110b
        sahf
```

См. также: уроки 2, 7 и команду lahf

SAL

(Shift Arithmetic operand Left)
Сдвиг арифметический операнда влево

Схема команды: sal операнд,количество_сдвигов
Назначение: арифметический сдвиг операнда влево.

Синтаксис
Алгоритм работы:

- сдвиг всех битов операнда влево на один разряд, при этом выдвигаемый слева бит становится значением флага переноса cf;
- одновременно справа в операнд вдвигается нулевой бит;
- указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Применение:
Команда sal используется для сдвига разрядов операнда влево. Так же, как и для других сдвигов, значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов количество_разрядов. Аналогично другим командам сдвига сохраняется эффект, связанный с поведением флага of, значение которого имеет смысл только в операциях сдвига на один разряд:

- если of=1, то текущее значение флага cf и выдвигаемого слева бита операнда различны;
- если of=0, то текущее значение флага cf и выдвигаемого слева бита операнда совпадают.

Этот эффект, как вы помните, обусловлен тем, что флаг cf устанавливается в единицу всякий раз при изменении знакового разряда операнда.
Команду sal удобно использовать для умножения целочисленных операндов без знака на степени 2. Кстати сказать, это самый быстрый способ такого умножения; умножить содержимое ax на 16 (2 в степени 4):

```
        mov     ax,17
        sal     ax,4
```

См. также: уроки 8, 9 и команды rcr, rcl, ror, rol, sar, shl, shr

SAR

(Shift Arithmetic operand Right)
Сдвиг арифметический операнда вправо

Схема команды: sar операнд,количество_сдвигов
Назначение: арифметический сдвиг операнда вправо.

Синтаксис
Алгоритм работы:

- сдвиг всех битов операнда вправо на один разряд, при этом выдвигаемый справа бит становится значением флага переноса cf;
- обратите внимание: одновременно слева в операнд вдвигается не нулевой бит, а значение старшего бита операнда, то есть по мере сдвига вправо освобождающиеся места заполняются значением знакового разряда. По этой причине этот тип сдвига и называется арифметическим;
- указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Применение:
Команда sar используется для арифметического сдвига разрядов операнда вправо. Так же, как и для других сдвигов, значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда количество_разрядов. В отличие от других команд сдвига флаг of всегда сбрасывается в ноль в операциях сдвига на один разряд.
Команду sar можно использовать для деления целочисленных операндов со знаком на степени 2.

```
        mov     ax,88
;(ax) разделить на 2 во второй степени,
то есть на 4
        sar     ax,2
```

См. также: урок 8, 9 и команды rcr, rcl, ror, rol, sal, shl, shr

SBB

(SuBtract with Borrow)
Вычитание с заемом

Схема команды: sbb операнд_1,операнд_2
Назначение: целочисленное вычитание с учетом результата предыдущего вычитания командами sbb и sub (по состоянию флага переноса cf).

Синтаксис
Алгоритм работы:

- выполнить сложение операнд_2=операнд_2+(cf);
- выполнить вычитание операнд_1=операнд_1-операнд_2;

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команда sbb используется для выполнения вычитания старших частей значений многобайтных операндов с учетом возможного предыдущего заема при вычитании младших частей значений этих операндов.

```
;выполнить вычитание 64-битных
значений: vich_1-vich_2
vich_1  dd      2 dup (0)
vich_2  dd      2 dup (0)
rez     dd      2 dup (0)
...
;ввести значения в поля vich_1 и
vich_2:
;младший байт по младшему адресу
...
        mov     eax,vich_1
        sub     eax,vich_2
;вычесть младшие половинки чисел
        mov     rez,eax ;младшая часть
результата
        mov     eax,vich_1+4
        sbb     eax,vich_2+4
;вычесть старшие половинки чисел
        mov     rez+4,eax
;старшая часть результата
```

См. также: урок 8, Приложение 7 и команды sub

SCAS/SCASB/SCASW/SCASD

Сканирование строки байтов/слов/двойных слов
ASCII-коррекция после сложения

Схема команды:

scas приемник
scasb
scasw
scasd

Назначение: поиск значения в последовательности (цепочке) элементов в памяти.

Синтаксис
Алгоритм работы:

- выполнить вычитание (элемент цепочки-(eax/ax/al)). Элемент цепочки локализуется парой es:edi/di. Замена сегмента es не допускается;
- по результату вычитания установить флаги;

- изменить значение регистра edi/di на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df:
  - df=0 — величина положительная, то есть просмотр от начала цепочки к ее концу;
  - df=1 — величина отрицательная, то есть просмотр от конца цепочки к ее началу.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команды сканирования сравнивают значение в регистре eax/ax/al с ячейкой памяти, локализуемой парой регистров es:edi/di. Размер сравниваемого элемента зависит от применяемой команды. Команда scas может работать с элементами размером в байт, слово или двойное слово. В качестве операнда в команде указывается идентификатор последовательности элементов в памяти. Реально этот идентификатор используется лишь для получения типа элементов последовательности, а ее адрес должен быть предварительно загружен в указанную выше пару регистров. Транслятор, обработав команду scas и выяснив тип операндов, генерирует одну из машинных команд:: scasb, scasw или scasd. Машинного аналога для команды scas нет. Для адресации операнда источник обязательно должен использоваться регистр es.
Для того чтобы эту команду можно было использовать для поиска значения в последовательности элементов, имеющих размерность байт, слово или двойное слово, необходимо использовать один из префиксов repe или repne. Эти префиксы не только заставляют циклически выполняться команду поиска, пока *ecx/cx*<>0, но и отслеживают состояние флага zf (см. команды rep/repe/repne).

```
;сосчитать число пробелов в строке str
.data
str     db      '...'
len_str=$-str
.code
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        lea     di,str
        mov     cx,len_str      ;длину
строки – в cx
        mov     al,' '
        mov     bx,0    ;счетчик для
подсчета пробелов в строке
        cld
cycl:
repe    scasb
        jcxz    exit    ;переход на
exit, если цепочка просмотрена
полностью
        inc     bx
        jmp     cycl
exit:   ...
```

См. также: урок 11 и команды cmps/cmpsb/cmpsw/cmpsd, ins/insb/insw/insd, lods/lodsb/lodsw/lodsd, movs/movsb/movsw/movsd, outs, stos/stosb /stosw/stosd, rep/repe/repz/repne/repnz

SETcc

(byte SET on condition)
Установка байта по условию

Схема команды: setcc операнд
Назначение: установка операнда логическим значением в зависимости от истинности условия, заданного модификатором кода операции cc.

Синтаксис
Алгоритм работы:
Команда проверяет истинность условия, заданного в коде операции, то есть, фактически, состояние определенных флагов.

Команды установки байтов

Команда	Проверяемые флаги	Логическое условие
SETA/SETNBE	CF = 0 и ZF = 0	(выше)/(не ниже или равно)
SETAE/SETNB	CF = 0	(выше или равно)/(не ниже)
SETB/SETNAE	CF = 1	(ниже)/(не выше или равно)
SETBE/SETNA	CF = 1 или ZF = 1	(ниже или равно)/(не выше)
SETC	CF = 1	перенос
SETE/SETZ	ZF = 1	ноль
SETG/SETNLE	ZF = 0 или SF = OF	(больше)/(не меньше или равно)
SETGE/SETNL	SF = OF	(больше или равно)/(не меньше)
SETL/SETNGE	SF <> OF	если SF <> OF
SETLE/SETNG	ZF=1 или SF <> OF	(меньше или равно)/(не больше)
SETNC	CF = 0	нет переноса
SETNE/SETNZ	ZF = 0	не равно нулю
SETNO	OF=0	нет переполнения
SETNP/SETPO	PF = 0	(неравенство)/(нет контроля четности)
SETNS	SF = 0	нет знака, число положительное
SETO	OF = 1	переполнение
SETP/SETPE	PF = 1	контроль четности/равенство
SETS	SF = 1	если знак минус, число отрицательное

Если проверяемое условие (или содержимое соответствующих флагов на момент выдачи команды setcc) истинно, то установить значение операнда в 01h, если условие ложно — то в 00h.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Эти команды можно использовать после любой команды, изменяющей флаги, при

необходимости анализа результата изменений. Если проанализировать условия для команд условного перехода, то обнаружится их полное соответствие с условиями, обрабатываемыми командой setcc, за исключением, конечно, команд jcxz и jecxz.

```
;подсчитать число единичных битов в
регистре ax
        mov     cx,16
m1:     rol     ax,1
        setc    bl
        add     bh,bl
        clc
        loop    m1
```

См. также: урок 10 и команду jcc

SGDT

(Store Global Descriptor Table)
Сохранение регистра глобальной дескрипторной таблицы

Схема команды: sgdt источник
Назначение: извлечение содержимого системного регистра gdtr, содержащего значения базового адреса и размера глобальной дескрипторной таблицы GDT.

Синтаксис
Алгоритм работы:
Команда выполняет чтение содержимого системного регистра gdtr в область памяти размером 48 бит. Структурно эти 48 бит представляют 16 бит размера и 32 бита значения базового адреса начала таблицы GDT в памяти.
Состояние флагов после выполнения команды:

```
выполнение команды не влияет на флаги
```

Применение:
Команду sgdt применяют при работе системных программ с уровнем привилегий 0, в частности, при написании различных драйверов.

```
.286
;структура для описания
псевдодескриптора gdtr
point   STRUC
lim     dw      0
adr     dd      0
        ENDS
.data
point_gdt       point
.code
...
;читаем содержимое gdtr
        sgdt    point_gdt
...
```

См. также: уроки 16, 17 и команду lgdt

SIDT

(Store Interrupt Descriptor Table)
Сохранение регистра глобальной дескрипторной таблицы прерываний

Схема команды: sidt источник
Назначение: извлечение содержимого системного регистра idtr, содержащего значения базового адреса и размера дескрипторной таблицы прерываний IDT.

Синтаксис
Алгоритм работы:
команда sidt выполняет чтение содержимого системного регистра idtr в область памяти размером 48 бит. Структурно эти 48 бит представляют 16 бит размера и 32 бита значения базового адреса начала таблицы IDT в памяти.
Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команду sidt применяют при работе системных программ с уровнем привилегий 0, в частности, при написании различных драйверов. В качестве операнда в команде указывается адрес области в формате 16+32. Младшее слово области — размер IDT, двойное слово по старшему адресу — значение базового адреса начала этой таблицы.

```
.286
;структура для описания
псевдодескрипторов gdtr и idtr
point   STRUC
lim     dw      0
adr     dd      0
        ENDS
.data
point_idt       point
.code
...
;читаем содержимое idtr
        sidt    point_idt
...
```

См. также: урок 17 и команду lidt

SHL

(SHift logical Left)
Сдвиг логический операнда влево

Схема команды: shl операнд,количество_сдвигов
Назначение: логический сдвиг операнда влево.

Синтаксис
Алгоритм работы:

- сдвиг всех битов операнда влево на один разряд, при этом выдвигаемый слева бит становится значением флага переноса cf;
- одновременно слева в операнд вдвигается нулевой бит;
- указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Применение:
Команда shl используется для сдвига разрядов операнда влево. Ее машинный код идентичен коду sal, поэтому вся информация, приведенная для sal, относится и к команде shl. Команда shl используется для сдвига разрядов операнда влево. Так же, как и для других сдвигов, значение второго операнда (счетчикк сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда количество_разрядов. Аналогично другим командам сдвига сохраняется эффект, связанный с поведением флага of, значение которого имеет смысл только в операциях сдвига на один разряд:

- если of=1, то текущее значение флага cf и выдвигаемого слева бита операнда различны;
- если of=0, то текущее значение флага cf и выдвигаемого слева бита операнда совпадают.

Этот эффект, как вы помните, обусловлен тем, что флаг of устанавливается в единицу всякий раз при изменении знакового разряда операнда.
Команду shl удобно использовать для умножения целочисленных операндов без знака на степени 2. Кстати сказать, это самый быстрый способ умножения; умножить содержимое ax на 16 (2 в степени 4).

```
        mov     ax,17
        shl     ax,4
```

См. также: урок 9 и команды rcr, rcl, ror, rol, sar, sal, shr

SHLD

(SHift Left Double word)
Сдвиг двойного слова влево

Схема команды:
shld
приемник,источник,количество_сдвигов

Назначение: логический сдвиг двойного слова влево.

Синтаксис
Алгоритм работы:

- сдвинуть операнд приемник влево на количество битов, определяемое операндом количество_сдвигов;

- одновременно сдвинуть операнд источник влево на количество битов, определяемое операндом количество_сдвигов. Важно заметить, что операнд источник только обеспечивает вдвигаемые в операнд приемник биты, сам он при этом не изменяется;
- выдвигаемые во время сдвига влево из операнда источник биты вдвигаются в операнд приемник с его правого края.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	r	r	?	r	r

Применение:
Команда shld используется для манипуляции битовыми строками длиной до 64 бит. Эту команду удобно использовать для быстрой вставки (или извлечения) битной строки в большую битную строку; при этом, что очень важно, не разрушается контекст (битное окружение) этих подстрок.

```
.386
;извлечь старшую половину eax в bx без
разрушения eax
        mov     cl,16
        shld    ebx,eax,cl
        push    bx
        shl     ebx,cl
        shld    eax,ebx,cl
;восстановим eax pop bx
```

См. также: урок 9 и команды rcr, rcl, ror, rol, sar, sal, shr, shrd

SHR

Сдвиг логический операнда вправо
ASCII-коррекция после сложения

Схема команды: shr операнд,кол-во_сдвигов
Назначение: логический сдвиг операнда вправо.

Синтаксис
Алгоритм работы:

- сдвиг всех битов операнда вправо на один разряд; при этом выдвигаемый справа бит становится значением флага переноса cf;
- одновременно слева в операнд вдвигается нулевой бит;
- указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF

?r	r	r	?	r	r

Применение:
Команда shr используется для логического сдвига разрядов операнда вправо. Так же, как и для других сдвигов, значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда количество_разрядов. В отличие от других команд сдвига, флаг of всегда сбрасывается в ноль в операциях сдвига на один разряд.
Команду shr можно использовать для деления целочисленных операндов без знака на степени 2.

```
        mov     cl,4
        shr     eax,cl  ;(eax)
разделить на 2 в степени 4
```

См. также: урок 9 и команды rcr, rcl, ror, rol, sal, shl, sar

SHRD

(SHift Right Double word)
Сдвиг двойного слова вправо

Схема команды: shrd приемник,источник,количество_сдвигов

Назначение: логический сдвиг двойного слова вправо.

Синтаксис
Алгоритм работы:

- сдвинуть операнд приемник вправо на количество битов, определяемое операндом количество_сдвигов;
- одновременно сдвинуть операнд источник вправо на количество битов, определяемое операндом количество_сдвигов. Важно заметить, что операнд источник только обеспечивает вдвигаемые в операнд приемник биты, сам он при этом не изменяется;
- выдвигаемые вправо во время сдвига из операнда источник биты вдвигаются в операнд приемник с его левого конца.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	r	r	?	r	r

Применение:
Команда shrd используется для манипуляции битными строками длиной до 64 бит. Эту команду удобно использовать для быстрой вставки (или извлечения) битной строки в большую битную строку, при этом, что очень важно, не разрушается контекст (битное окружение) этих подстрок.

```
.386
;разделить операнд размером 64 бит на
```

```
степень 2
op_l    dd      ...     ;младшая часть
операнда
op_h    dd      ...     ;старшая часть
операнда
...
        mov     eax,op_h
        shrd    op_l,eax,4
;разделить операнд на 4
;так как старшая часть операнда реально
еще не сдвинулась,
;то нужно привести ее в соответствие с
результатом
        shr     op_h,4
```

См. также: урок 9 и команды rcr, rcl, ror, rol, sar, sal, shr, shld

STC

(Set Carry Flag)
Установка флага переноса

Схема команды: stc
Назначение: установка флага переноса cf в 1.

Синтаксис
Алгоритм работы:
установить флаг cf в единицу.

Состояние флагов после выполнения команды:

00
CF
1

Применение:
Данная команда используется для установки флага cf в единицу. Такая необходимость может возникнуть при работе с командами сдвига, арифметическими командами или действиями по индикации ошибок в программах.

```
...
        stc             ;cf=1
...
```

См. также: уроки 2, 8, 9 и команды cmc, clc

STD

(SeT Direction Flag)
Установка флага направления

Схема команды: std

Назначение: установка флага направления df в 1.

Синтаксис
Алгоритм работы:
установить флаг df в единицу.

Состояние флагов после выполнения команды:

10
DF
1

Применение:
Данная команда используется для установки флага df в единицу. Такая необходимость может возникнуть при работе с цепочечными командами. Единичное состояние флага df вынуждает микропроцессор производить декремент регистров si и di при выполнении цепочечных операций.

```
...
        std             ;df=1
;смотрите материал урока 11
```

См. также: уроки 2, 11 и команду cld

STI

(SeT Interrupt flag)
Установка флага прерывания

Схема команды: sti

Назначение: установка флага прерывания if в единицу.

Синтаксис
Алгоритм работы:
установить флаг if в единицу.

Состояние флагов после выполнения команды:

09
IF
1

Применение:
Данная команда используется для установки флага if в единицу. Такая необходимость может возникнуть при разработке программ обработки прерываний.

```
...
        sti             ;if=1
```

См. также: урок 2, 15, 17 и команду cli

STOS/STOSB/STOSW/STOSD

(Store String Byte/Word/Double word operands)
Сохранение строки байтов/слов/двойных слов

Схема команды:

stos приемник
stosb
stosw
stosd

Назначение: сохранение элемента из регистра-аккумулятора al/ax/eax в последовательности (цепочке).

Синтаксис
Алгоритм работы:

- записать элемент из регистра al/ax/eax в ячейку памяти, адресуемую парой es:di/edi. Размер элемента определяется неявно (для команды stos) или конкретной применяемой командой (для команд stosb, stosw, stosd);
- изменить значение регистра di на величину, равную длине элемента цепочки. Знак этого изменения зависит от состояния флага df:
  - df=0 — увеличить, что означает просмотр от начала цепочки к ее концу;
  - df=1 — уменьшить, что означает просмотр от конца цепочки к ее началу.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команды сохраняют элемент из регистров al/ax/eax в ячейку памяти. Перед командой stos можно указать префикс повторения rep, в этом случае появляется возможность работы с блоками памяти, заполняя их значениями в соответствии с содержимым регистра *ecx/cx*.

```
;заполнить некоторую область памяти
пробелами
str     db      'Какая-то строка'
len_str=$-str
...
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        cld
        mov     al,' '
        lea     di,str
        mov     cx,len_str
rep     stosb           ;заполняем
пробелами строку str
```

```
;пример совместной работы stosb и
lodsb:
;копировать одну строку в другую до
первого пробела
```

```
str1    db      'Какая-то строка'
len_str1=$-str
str2    db      len_str1 dup (' ')
...
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        cld
        mov     cx,len_str1
        lea     si,str1
        lea     di,str2
m1:     lodsb
        cmp     al,' '
        jc      exit    ;выход, если
пробел
        stosb
        loop    m1
exit:
```

См. также: урок 11 и команды ins/insb/insw/insd, cmps/cmpsb/cmpsw/cmpsd, movs/movsb/movsw/movsd, outs, scas/scasb/scasw/scasd, lods/lodsb/lodsw/lodsd, rep/repe/repz/repne/repnz

SUB

(SUBtract)
Вычитание

Схема команды: sub операнд_1,операнд_2
Назначение: целочисленное вычитание.

Синтаксис
Алгоритм работы:

- выполнить вычитание операнд_1=операнд_2-операнд_1;
- установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команда sub используется для выполнения вычитания целочисленных операндов или для вычитания младших частей значений многобайтных операндов.

```
;выполнить вычитание 64-битных
значений: vich_1-vich_2
vich_1  dd      2 dup (0)
vich_2  dd      2 dup (0)
rez     dd      2 dup (0)
...
;ввести значения в поля vich_1 и
vich_2:
```

```
;младший байт по младшему адресу
...
        mov     eax,vich_1
        sub     eax,vich_2
;вычесть младшие половинки чисел
        mov     rez,eax ;младшая часть
результата
        mov     eax,vich_1+4
        sbb     eax,vich_2+4
;вычесть старшие половинки чисел
        mov     rez+4,eax
;старшая часть результата
```

См. также: урок 8, приложение 7 и команду sbb

TEST

(TEST operand)
Логическое И

Схема команды: test приемник,источник

Назначение: операция логического сравнения операндов приемник и источник размерностью байт, слово или двойное слово.

Синтаксис

Алгоритм работы:

- выполнить операцию логического умножения над операндами приемник и источник: бит результата равен 1, если соответствующие биты операндов равны 1, в остальных случаях бит результата равен 0;
- установить флаги.

Состояние флагов после выполнения команды:

11	07	06	02	00
OF	SF	ZF	PF	CF
0	r	r	r	0

Применение:
Команда test используется для логического умножения двух операндов. Результат операции, в отличие от команды and, никуда не записывается, устанавливаются только флаги. Эту команду удобно использовать для получения информации о состоянии заданных битов операнда приемник. Для анализа результата используется флаг zf, который равен 1, если результат логического умножения равен нулю.

```
        test    al,01h
        jnz     m1      ;переход, если
нулевой бит al равен 1
```

См. также: урок 9 и команды or, xor, and, bt

XADD

(eXchange and ADD)
Обмен и сложение

Схема команды: xadd приемник,источник
Назначение: суммирование и обмен двух значений.

Синтаксис
Алгоритм работы:

- копировать содержимое операнда приемник в операнд источник;
- выполнить сложение (приемник+источник);
- поместить сумму в операнд приемник.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение:
Команда xadd используется для выполнения операции обмена и сложения двух операндов.

```
        mov     al,08h
        mov     bl,01h
        xadd    al,bl   ;al=09h, bl=08h
```

См. также: уроки 7, 8 и команды add, xchg

XCHG

(eXCHanGe)
Обмен

Схема команды: xchg операнд_1,операнд_2
Назначение: обмен двух значений между регистрами или между регистрами и памятью.

Синтаксис
Алгоритм работы:
обмен содержимого операнд_1 и операнд_2.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команду xchg можно использовать для выполнения операции обмена двух операндов с целью изменения порядка следования байт, слов, двойных слов или их временного сохранения в регистре или памяти. Альтернативой является использование для этой цели стека.

```
;поменять порядок следования байт в
```

```
слове
ch1     label   byte
        dw      0f85ch
...
        mov     al,ch1
        xchg    ch1+1,al
        mov     ch1,al
```

См. также: урок 7 и команды bswap, cmpxchg, xadd

XLAT/XLATB

(transLATe Byte from table)
Преобразование байта

Схема команды: xlat адрес_таблицы_байтов xlatb
Назначение: подмена байта в регистре al байтом из последовательности (таблицы) байтов в памяти.

Синтаксис
Алгоритм работы:

- вычислить адрес, равный ds:bx+(al);
- выполнить замену байта в регистре al байтом из памяти по вычисленному адресу.

Несмотря на наличие операнда адрес_таблицы_байтов в команде xlat, адрес последовательности байтов, из которой будет осуществляться выборка байта для подмены в регистре al, должен быть предварительно загружен в пару ds:bx(ebx). Команда xlat допускает замену сегмента.

Состояние флагов после выполнения команды:

выполнение команды не влияет на флаги

Применение:
Команду xlat можно использовать для выполнения перекодировок символов. Для формирования адреса таблицы в регистрах bx(ebx) можно использовать команду lea или оператор ассемблера offset в команде mov.

```
table   db      'abcdef'
int     db      0       ;значение
индекса
...
        mov     al,3
        lea     bx,table
        xlat            ;(al)='c'
```

См. также: урок 7 и команду lea

XOR

Логическое исключающее ИЛИ

ASCII-коррекция после сложения

Схема команды: xor приемник,источник
Назначение: операция логического исключающего ИЛИ над двумя операндами размерностью байт, слово или двойное слово.

Синтаксис
Алгоритм работы:

- выполнить операцию логического исключающего ИЛИ над операндами: бит результата равен 1, если значения соответствующих битов операндов различны, в остальных случаях бит результата равен 0;
- записать результат сложения в приемник;
- установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
0	r	r	?	r	0

Применение:
Команда xor используется для выполнения операции логического исключающего ИЛИ двух операндов. Результат операции помещается в первый операнд. Эту операцию удобно использовать для инвертирования или сравнения определенных битов операндов.

```
;изменить значение бита 0 регистра al
на обратное
        xor     al,01h
```

См. также: урок 9 и команды and, or, not