Машинно-ориентированное программирование

лектор – Скороход Сергей Васильевич

Учебная карта дисциплины

№	Виды контрольных мероприятий	Тек. контроль	Рубежный контроль *(при наличии)*
	Модуль 1.	**50**	
1	Лабораторная работа 1	6	
2	Лабораторная работа 2	8	
3	Лабораторная работа 3	8	
4	Лабораторная работа 4	8	
5	Письменный контрольный опрос по теории		20
	Модуль 2.	**50**	
1	Лабораторная работа 5	10	
2	Лабораторная работа 6	10	
3	Лабораторная работа 7	10	
4	Письменный контрольный опрос по теории		20
	Бонусные баллы	**10**	За посещение лекций – 7 За посещение лаб. работ - 3
	Промежуточная аттестация в форме диф. зачета		

Источники

Тематический сайт по дисциплине: https://assembler-mop.nethouse.ru httphttp://assembler-mop.http://assembler-mop.mopevmhttp://assembler-mop.mopevm.http://assembler-mop.mopevm.sfeduhttp://assembler-mop.mopevm.sfedu.ru

Литература

Любая литература по языку Ассемблера для процессоров Intel семейства x86

- Учебное пособие 2016 (коллектив авторов МОП ЭВМ),
- Магда,
- Юров,
- Голубь,
- Зубков,
- Пирогов
- и др доступные в Интернет книги.

График сдач лабораторных работ

Для группы КТбо2-6

Неделя	1	2	3	4	5	6	7	8	9-НАМ	10	11	12	13	14	15	16	17
	07.09	14.09	21.09	28.09	05.10	12.10	19.10	26.10	02.11	09.11	16.11	23.11	30.11	07.12	14.12	21.12	28.12
Лаб. раб.1	X	100	100	80	80	60	60	60	60	60	60	60	60	60	60	60	60
Лаб. раб.2			X	100	100	80	80	60	60	60	60	60	60	60	60	60	60
Лаб. раб.3					X	100	100	80	80	60	60	60	60	60	60	60	60
Лаб. раб.4							X	100	100	80	80	60	60	60	60	60	60
Лаб. раб.5									X	100	100	80	80	60	60	60	60
Лаб. раб.6												X	100	100	80	80	60
Лаб. раб.7														X	100	100	80

Л. Рабо	Максимальный балл
1	6
2	8
3	8
4	8
5	10
6	10
7	10

СИСТЕМЫ СЧИСЛЕНИЯ

Система счисления (СС)- способ представления (записи) чисел с помощью некоторых символов (цифр)

Непозиционная система счисления - вес цифры не зависит от позиции, которую она занимает в числе.

Пример – римская СС: XIV = 10 + 5 -1

Позиционная система счисления - вес каждой цифры изменяется в зависимости от ее позиции в записи числа.

Пример – десятичная СС:

111 = 100 + 10 + 1

Позиционные системы счисления

- ***Вес цифры*** – определяется ее положением в записи числа.
- ***Основание СС*** – количество цифр, используемых для записи числа.
- ***Разряд***– позиция цифры в числе. Разряды нумеруются с 0, справа налево.

Развернутая форма записи числа:

$$A_q = a_{n-1}q^{n-1} + \ldots + a_1q^1 + a_0q^0 + a_{-1}q^{-1} + \ldots + a_{-m}q^{-m}$$

A_q — число в системе счисления с основанием q,

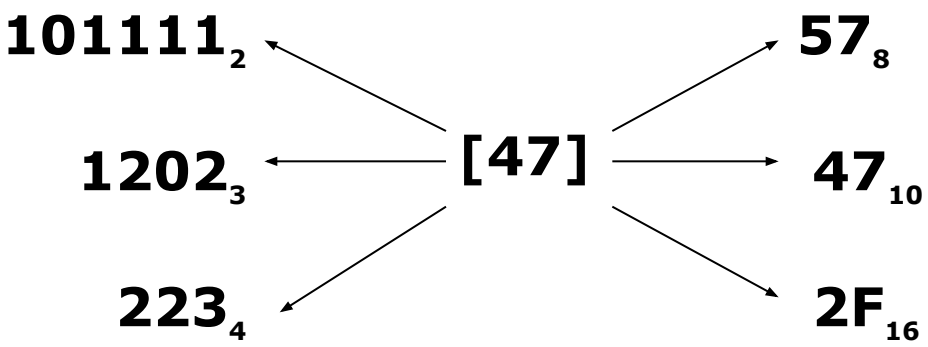
q — основание системы счисления (*количество используемых цифр*),

a — цифры многоразрядного числа A_q,

n (m) — количество целых (дробных) разрядов числа A_q .

ПРЕДСТАВЛЕНИЕ ЧИСЛА

Одно и то же число может быть представлено в различных системах счисления (с разными основаниями)



[.] – значение числа. Для простоты понимания значение числа всегда указывается в десятичной СС

ПЕРЕВОД ЧИСЛА $N_p \Rightarrow N_q$

Перевод числа из одной СС в другую осуществляется в два этапа:

1) переводится целая часть числа;
2) переводится дробная часть числа.

ПЕРЕВОД ЧИСЛА $N_p \Rightarrow N_q$.

(правило перевода целой части числа)

Для перевода целого числа N_p в число N_q необходимо N_p делить на основание q (по правилам, принятым в $СС_p$) до получения целого остатка, меньшего q. Полученное частное снова необходимо делить на основание q до получения целого остатка, меньшего q и т.д. до тех пор, пока последнее частное не будет меньше q.

Число N_q представится в виде упорядоченной последовательности цифр $СС_q$ (остатков от деления) в порядке, обратном получению, причем старшую цифру числа N_q даст последнее частное

ПЕРЕВОД ЧИСЛА $N_p \Rightarrow N_q$.

(правило перевода дробной части)

Перевод правильной дроби N_p в число N_q заключается в последовательном умножении дроби N_p на основание q (по правилам, принятым в $СС_p$), причем перемножению подвергается только дробная часть.

Дробь N_q представится в виде упорядоченной последовательности целых частей произведений в порядке их получения.

В общем случае при переводе может возникать погрешность вследствие конечности разрядной сетки. Если требуемая точность перевода есть q^{-k}, то число указанных произведений должно быть равно k.

ПЕРЕВОД ЧИСЛА $N_p \Rightarrow N_q$.

(упражнения)

$349_{10} \rightarrow ?_4$

$0{,}41_{10} \rightarrow ?_2$

$24{,}18_{10} \rightarrow ?_3$

$534_{10} \rightarrow ?_{16}$

$3A_{16} \rightarrow ?_4$

$2{,}7_8 \rightarrow ?_{10}$

$3{,}7_8 \rightarrow ?_2$

ПЕРЕВОД ЧИСЛА $N_8 \Rightarrow N_2$, $N_{16} \Rightarrow N_2$

- Перевод восьмиричных и шестнадцатиричных чисел в двоичную систему: каждую цифру заменить эквивалентной ей двоичной *триадой* (тройкой цифр) или *тетрадой* (четверкой цифр).

Примеры:

$5371_8 = 101\ 011\ 111\ 001_2$;

5 3 7 1

$1A3F_{16} = 1\ 1010\ 0011\ 1111_2$

1 A 3 F

ПЕРЕВОД ЧИСЛА $N_2 \Rightarrow N_8$, $N_2 \Rightarrow N_{16}$

Чтобы перевести число из <u>двоичной</u> системы в <u>восьмеричную</u> или <u>шестнадцатеричную</u>, его нужно разбить влево и вправо от запятой на *триады* (для восьмеричной) или *тетрады* (для шестнадцатеричной) и каждую такую группу заменить соответствующей восьмеричной (шестнадцатеричной) цифрой.

Примеры:

$1101010000111_2 = 1\quad 5\quad 2\quad 0\quad 7_8$;

1 101 010 000 111

$11011100001101_2 = 6\quad E\quad 0\quad D_{16}$

110 1110 0000 1101

ДВОИЧНАЯ АРИФМЕТИКА

Таблица сложения
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 10

Таблица вычитания
0 – 0 = 0
1 – 0 = 1
1 – 1 = 0
10 – 1 = 1

Таблица умножения
0 × 0 = 0
1 × 0 = 0
1 × 1 = 1

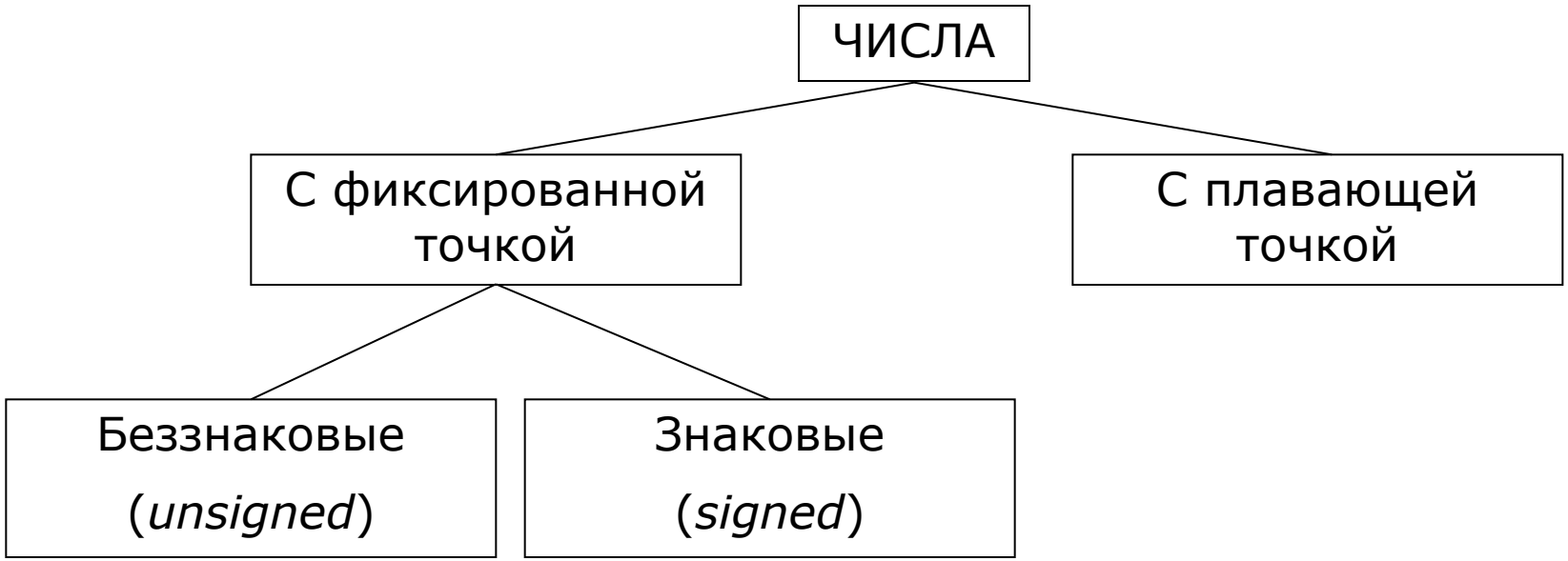
```
  1 1 0 1 1
+ 1 0 1 1 0 1
-------------
1 0 0 1 0 0 0
```

```
1 0 0 1 0 0 0
- 1 0 1 1 0 1
-------------
  1 1 0 1 1
```

```
1 1 0 1 0 1 0 0 1 | 1 0 0 0 1
1 0 0 0 1         | 1 1 0 0 1
  1 0 0 1 1
  1 0 0 0 1
      1 0 0 0 1
      1 0 0 0 1
      0 0 0 0 0
```

```
      1 1 0 0 1
    * 1 0 0 0 1
-----------------
      1 1 0 0 1
    0 0 0 0 0 _
  0 0 0 0 0 _ _
0 0 0 0 0 _ _ _
1 1 0 0 1 _ _ _ _
-----------------
1 1 0 1 0 1 0 0 1
```

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ В ЭВМ



Числа с плавающей точкой

Местоположение точки в записи числа изменяется
так, чтобы слева от точки оставался один разряд,
а смещение точки описывалось экспонентой:

$$\begin{array}{r} 4.8 * 10^{1} \\ 5.6 * 10^{1} \\ \hline 10.4 * 10^{1} \end{array} \rightarrow \mathbf{1.04 * 10^{2}}$$

Формат	*S*	*E*	*M*	*Диапазон чисел:*
Single (32)	1 (31)	8 (30 – 23)	23 (22-00)	от ± $\sim 10^{-44.85}$ до $\sim 10^{38.53}$
Double (64)	1 (63)	11 (62 - 52)	52 (51-00)	от ± $\sim 10^{-323.3}$ до $\sim 10^{308.3}$

Достоинства: большой диапазон обрабатываемых значений

Недостатки: сложность в реализации устройства обработки, ошибки округления

Числа с фиксированной точкой

- Местоположение точки в записи числа не изменяется.
Считается, что точка расположена справа от самого младшего разряда:

```
01011001.        48.
                 56.
               ------
                104.
```

Достоинства: простота реализации устройства обработки высокая точность, интуитивная понятность

Недостатки: малый диапазон возможных значений

ПРЕДСТАВЛЕНИЕ ЧИСЕЛ БЕЗ ЗНАКА

- Представление беззнакового (unsigned) числа соответствует его записи в заданной системе счисления (двоичной или шестнадцатеричной)

Машинное представление числа выполняется с учетом разрядности *n* машинного слова

Диапазон представления беззнаковых чисел:

$$0 \le x \le 2^n - 1$$

ПРЕДСТАВЛЕНИЕ ЗНАКОВЫХ ЧИСЕЛ

Знаковые (signed) числа представляются в ЭВМ:

- в ***прямом*** коде;
- в ***обратном*** коде;
- в ***дополнительном*** коде

Для обозначения знака числа выделяется специальный знаковый разряд, в котором записывается «0» для положительного числа и «1» для отрицательного числа.

Знаковый разряд всегда располагается слева от значащих разрядов (старший бит).

ПРЯМОЙ КОД

Число представляется в виде его абсолютного значения и кода знака

$$[x]_{\text{пр}} = \begin{cases} |x|, & \text{если } x \geq 0; \\ 2^{n-1} + |x|, & \text{если } x \leq 0 \end{cases}$$

Диапазон представления:

$$1-2^{n-1} \leq x \leq 2^{n-1} - 1$$

Представления «0»:

$$[+0]_{\text{пр}} = 0'00\ldots0_2$$

$$[-0]_{\text{пр}} = 1'00\ldots0_2$$

ПРЯМОЙ КОД (пример)

Представить в прямом коде для $n=5$, $n=8$

$x = [13]$ $x = [-13]$

$$/x/ = /13/_{10} = 1101_2 = D_{16}$$

<u>$n=5$:</u> $[13] = 0'1101_{пр,2}$	$[-13] = 1'1101_{пр,2}$
<u>$n=8$:</u> $[13] = 0'0001101_{пр,2} = 0D_{пр,16}$	$[-13] = 1'0001101_{пр,2} = 8D_{пр,2}$

ОБРАТНЫЙ КОД

Обратный код положительного числа $x \geq 0$ содержит «0» в старшем знаковом разряде и обычное представление x в остальных разрядах.

Если $x <= 0$, то знаковый разряд содержит «1», а остальные разряды содержат инвертированные значения

$$[x]_{\text{обр}} = \begin{cases} |x|, & \text{если } x \geq 0; \\ (2^{n-1} - 1) - |x|, & \text{если } x \leq 0 \end{cases}$$

Диапазон представления:

$$1\text{-}2^{n-1} \leq x \leq 2^{n-1} - 1$$

Представления «0»:

$$[+0]_{\text{обр}} = 0'00\ldots0_2$$

$$[-0]_{\text{обр}} = 1'11\ldots1_2$$

ОБРАТНЫЙ КОД (пример)

Представить в обратном коде для

$n=5$, $n=8$

$x = [13]$ $\qquad$ $x = [-13]$

$$/x/ = /13/_{10} = 1101_2 = D_{16}$$

$\underline{n=5:}$ $[13] = 0'1101_{пр,2} = 0'1101_{обр,2}$	$[-13] = 1'1101_{пр,2} = 1'0010_{обр,2}$
$\underline{n=8:}$ $[13] = 0'0001101_{пр,2} = 0D_{пр,16} = 0'0001101_{обр,2} = 0D_{обр,16}$	$[-13] = 1'0001101_{пр,2} = 8D_{пр,2} = 1'1110010_{обр,2} = F2_{обр,16}$

ПРАВИЛО СЛОЖЕНИЯ В ОБРАТНОМ КОДЕ

- Коды слагаемых суммируются, включая знаковый разряд, с циклическим (круговым) переносом.
- Результат верен, если не произошло переполнение.
- *Переполнение* происходит тогда, когда перенос в знаковый разряд (C_s) не равен переносу из знакового разряда (C_{s+1})

Пример. n=4

-5-2 = -7 (нет переполнения)

-5-4 = -9 (есть переполнение)

ДОПОЛНИТЕЛЬНЫЙ КОД

Дополнительный код положительного числа $x \geq 0$ содержит «0» в старшем знаковом разряде и обычное представление x в остальных разрядах (совпадает с прямым и обратным).

Если $x<0$, то знаковый разряд содержит «1», а остальные разряды содержат дополнение модуля исходного числа до 2^{n-1}.

$$[x]_{\text{доп}} = \begin{cases} |x|, & \text{если } x \geq 0; \\ 2^{n-1} - |x|, & \text{если } x < 0 \end{cases}$$

Диапазон представления:

$$-2^{n-1} \leq x \leq 2^{n-1} - 1$$

Представление «0»:

$$[0]_{\text{обр}} = 0'00\ldots0_2$$

ДОПОЛНИТЕЛЬНЫЙ КОД (пример)

Представить в дополнительном коде для $n=5$, $n=8$

$x = [13]$ $x = [-13]$

$$/x/ = /13/_{10} = 1101_2 = D_{16}$$

$\underline{n=5:}$ $[13] = 0'1101_{пр,2}$ $= 0'1101_{обр,2} =$ $= 0'1101_{доп,2}$	$[-13] = 1'1101_{пр,2} =$ $= 1'0010_{обр,2} =$ $= 1'0011_{доп,2}$
$\underline{n=8:}$ $[13] =$ $0'0001101_{пр,2} = 0D_{пр,16} =$ $= 0'0001101_{обр,2} = 0D_{обр,16}$ $= 0'0001101_{доп,2} = 0D_{доп,16}$	$[-13] = 1'0001101_{пр,2} =$ $8D_{пр,2} =$ $1'1110010_{обр,2} = F2_{обр,16}$ $= 1'1110011_{доп,2} = F3_{доп,16}$

ПРАВИЛО СЛОЖЕНИЯ В ДОПОЛНИТЕЛЬНОМ КОДЕ

- Коды слагаемых суммируются, включая знаковый разряд. Перенос (если он есть) отбрасывается.
- Результат верен, если не произошло переполнение.
- *Переполнение* происходит тогда, когда перенос в знаковый разряд (C_s) не равен переносу из знакового разряда (C_{s+1})

Пример. n=4

-5 + 7 = 2 (нет переполнения)

-5 – 7 - -12 (есть переполнение)

УВЕЛИЧЕНИЕ РАЗРЯДНОСТИ ЧИСЕЛ ПРИ ПРИСВАИВАНИИ

- *Для беззнаковых (unsigned) чисел* поле расширения в переменной-результате заполняется нулями
- *Для знаковых (signed) чисел* поле расширения в переменной-результате заполняется знаковым битом

В языках высокого уровня способ расширения выбирается и реализуется компилятором по типу данных автоматически

В Ассемблере программист самостоятельно выбирает способ реализации расширения разрядности переменной

УМНОЖЕНИЕ ЦЕЛОГО ЧИСЛА НА КОНСТАНТУ ПОСРЕДСТВОМ СДВИГОВ

- Сдвиг *беззнаковых* (*unsigned)* или *знаковых* (*signed)* числа **влево** на *n* двоичных разрядов приводит к его **умножению** на 2^n
- Если переменную *V* необходимо умножить на константу *C*, то константа *C* представляется в виде суммы степеней числа 2, а результат умножения записывается как сумма сдвигов числа *V* на показатели степеней.

***Пример**: V = V * 25;*

$C = 25 = 11001_2 = 16+8+1 = 2^4 + 2^3 + 2^0$

V =** V*(16+8+1) = V*16 + V*8 + V =* ***(V<<4) + (V<<3) + V

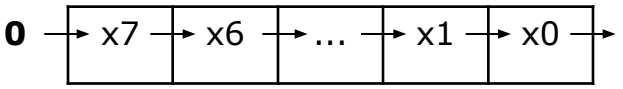
Достоинства: сдвиги и сложения выполняются быстрее, чем умножение

Недостатки: формула зависит от конкретного значения C, т.е. нельзя таким способом перемножить две переменных.

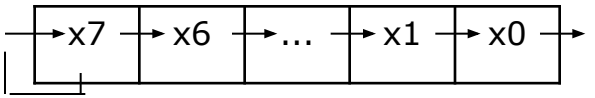
ДЕЛЕНИЕ ЦЕЛОГО ЧИСЛА НА 2^n ПОСРЕДСТВОМ СДВИГОВ

- Сдвиг *беззнаковых* (*unsigned)* или *знаковых* (*signed)* числа **вправо** на *n* двоичных разрядов приводит к его **делению** на 2^n

Для сдвига вправо *беззнаковых (unsigned)* чисел используется *логический* сдвиг:



Для сдвига вправо *знаковых (signed)* чисел используется *арифметический* сдвиг:



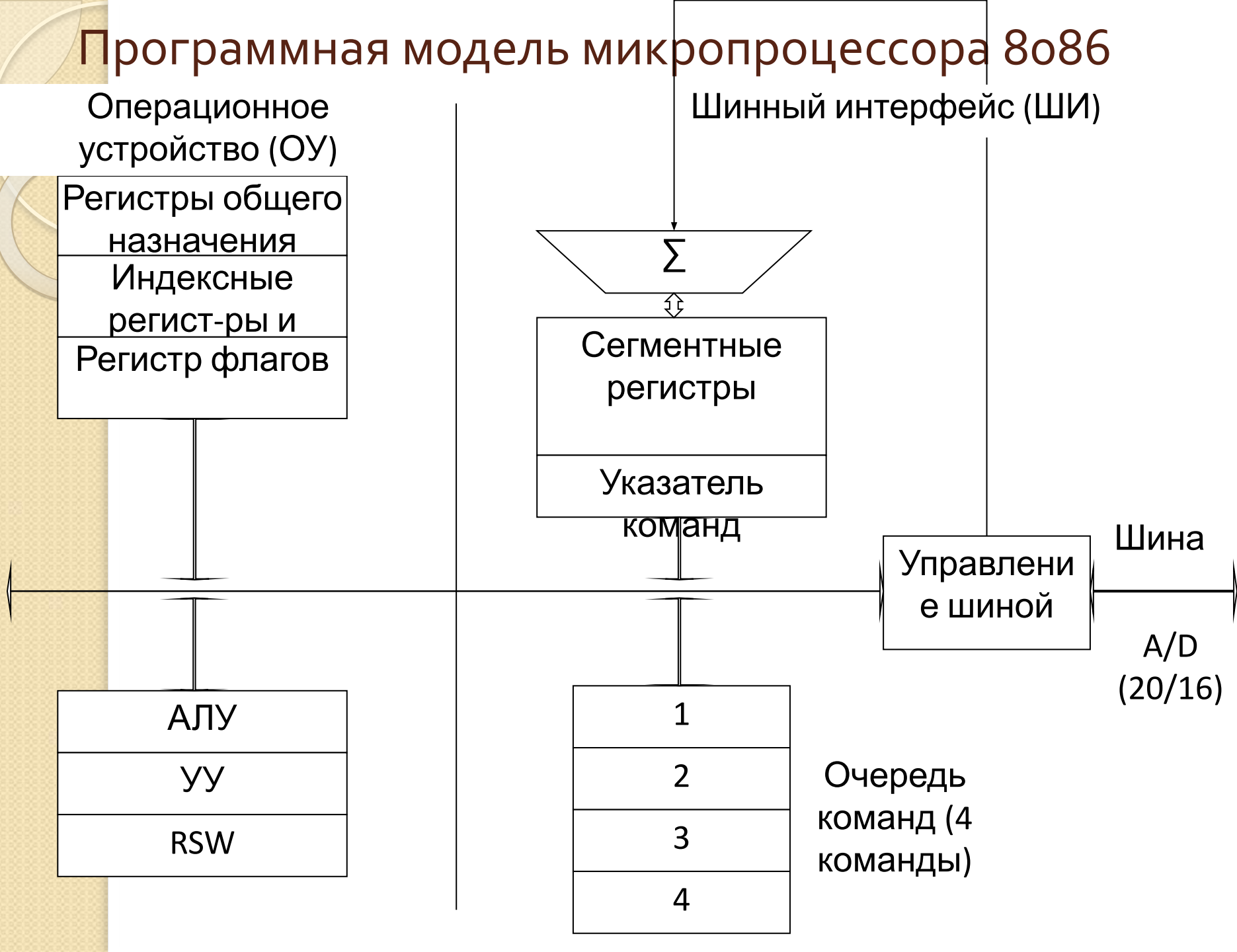
Пример: *V = V / 16;* C = 16 = 10000_2 = 2^4 ***V = (V>>4);***

Достоинства: сдвиги выполняются быстрее, чем деление. Компилятор автоматически выбирает правильную команду сдвига (по типу данных)

Недостатки: фактор сдвига зависит от конкретного значения *C*, таким способом нельзя выполнить деление на переменную или число ≠ 2^n

Микропроцессор Intel 8086

Программная модель микропроцессора 8086



Регистры процессора

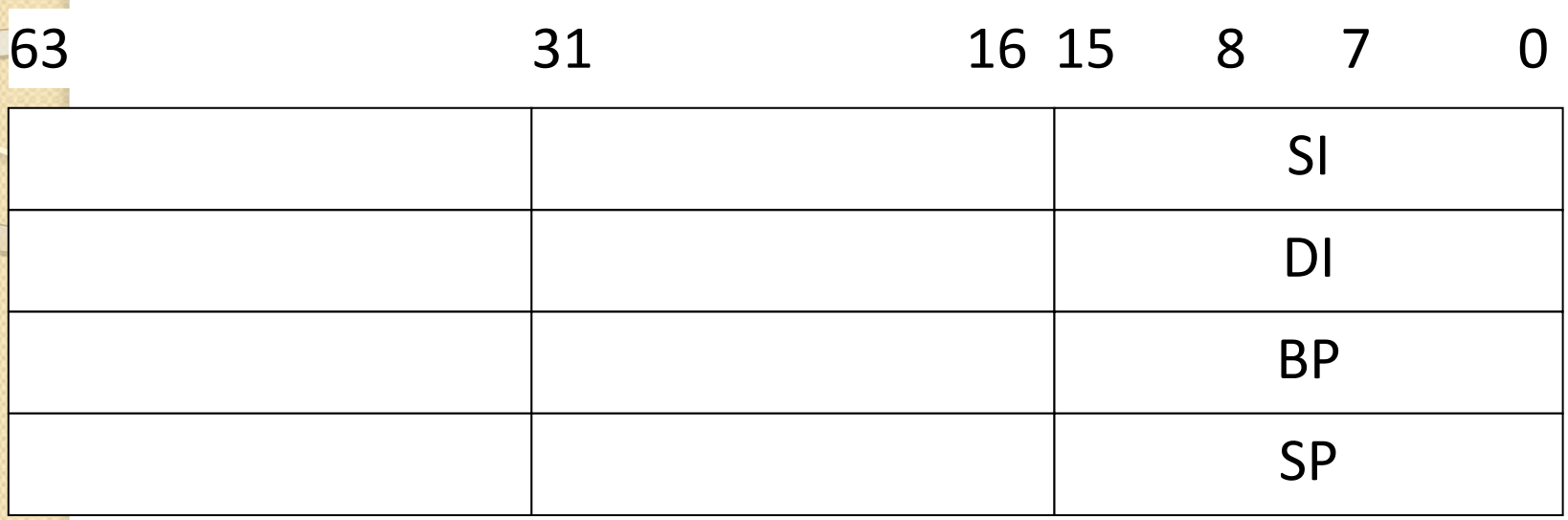
- Регистры общего назначения;
- Индексные регистры и указатели;
- Регистр флагов;
- Сегментные регистры;
- Указатель команд.

Регистры общего назначения

63 31	16 15 8	7 0
	AH	AL
	BH	BL
	CH	CL
	DH	DL

64 бита	32 бита	16 бит	8 бит
RAX	EAX	AX	AH/AL
RBX	EBX	BX	BH/BL
RCX	ECX	CX	CH/CL
RDX	EDX	DX	DH/DL

Индексные регистры и указатели



64 бита	32 бита	16 бит
RSI	ESI	SI
RDI	EDI	DI
RBP	EBP	BP
RSP	ESP	SP

Регистр флагов

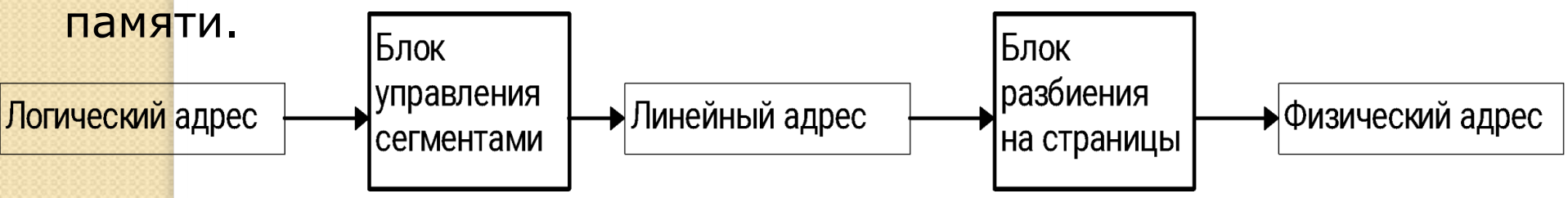
Бит №	*ИмяФлага*	*Назначение*
0	CF	Флаг переноса. Был ли перенос из или заем в старший разряд.
1	PF	Флаг четности. 1, если результат операции содержит четное количество единиц.
4	AF	Флаг вспомогательного переноса
6	ZF	Флаг нуля. 1, если результат операции равен 0.
7	SF	Флаг знака. Показывает знак результата операции (1- отрицательный, 0-положительный)
8	TF	Флаг трассировки. Обеспечивает возможность работы процессора в пошаговом режиме.
9	IF	Флаг внешних прерываний. Если IF=1, прерывание разрешается, IF=0 – блокируется.
10	DF	Флаг направления. Используется командами обработки строк. DF=1 – прямое направление (от меньших адресов к большим). DF=0 – обратное направление
12	OF	Флаг переполнения.

Сегментные регистры

Имя сегм. регистра	***Назначение***
CS	Регистр сегмента кода. Содержит начальный адрес сегмента кода.
DS	Регистр сегмента данных.
SS	Регистр сегмента стека.
ES	Дополнительный сегментный регистр.
FS, GS	Дополнительные сегментные регистры, появившиеся в более поздних поколениях процессоров

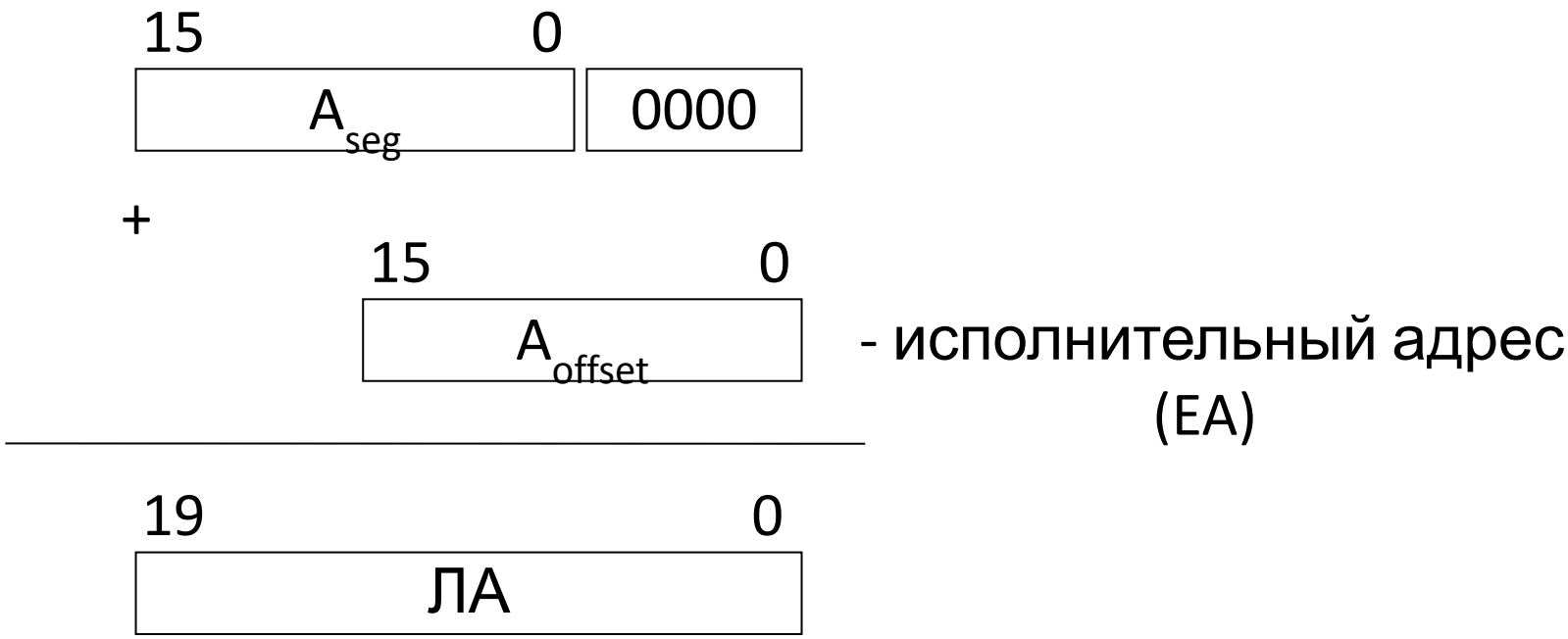
Организация памяти

- **Физическая память** – память на шине процессора.
- **Адресное пространство** – определяется разрядностью шины адреса.
- **Логический адрес** – адрес, используемый в программе.
- **Диспетчер памяти** – аппаратный механизм для доступа к памяти.

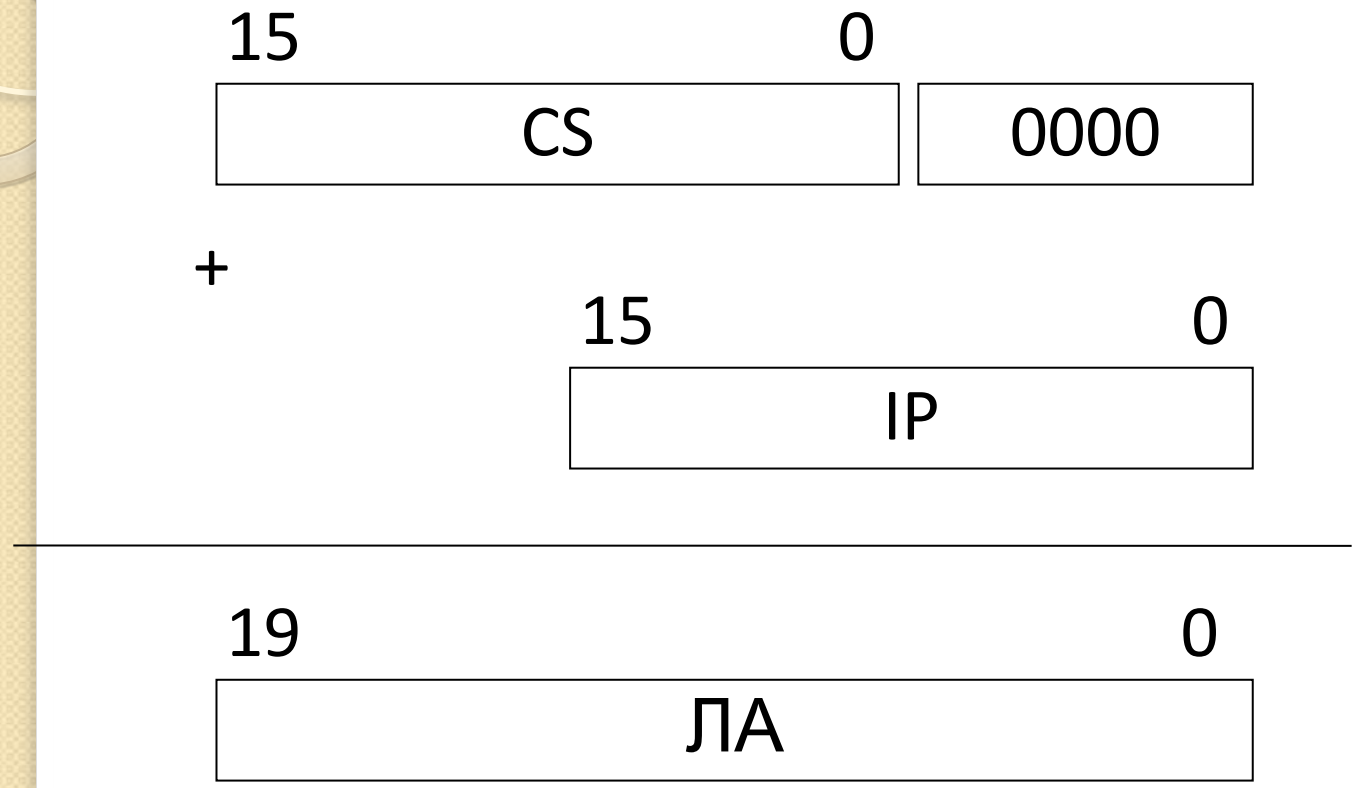


- Процессор может работать в 2 режимах: **реальном** и **защищенном**.
- Для программиста имеется 3 основных вида моделей памяти:
  - **Сегментная** модель реального режима.
  - **Сегментная** модель защищенного режима.
  - **Плоская** модель защищенного режима.

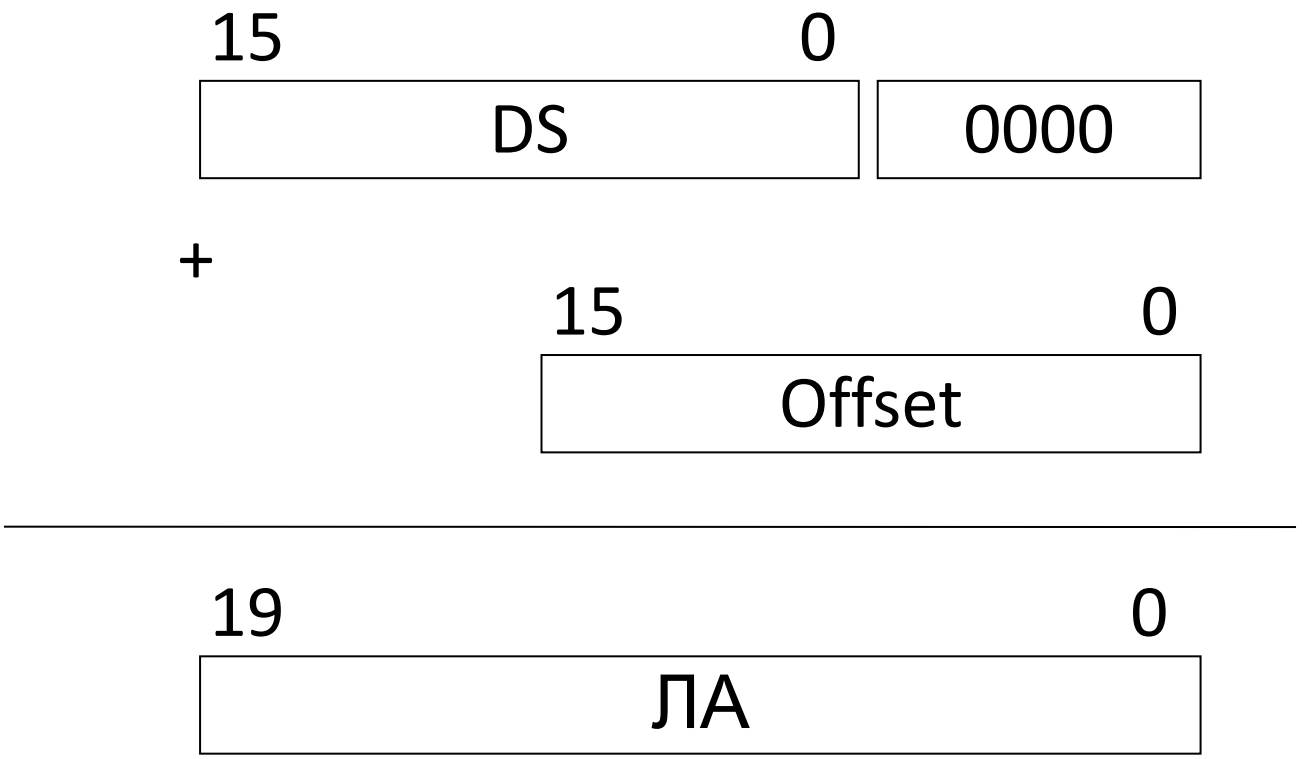
Сегментная адресация памяти



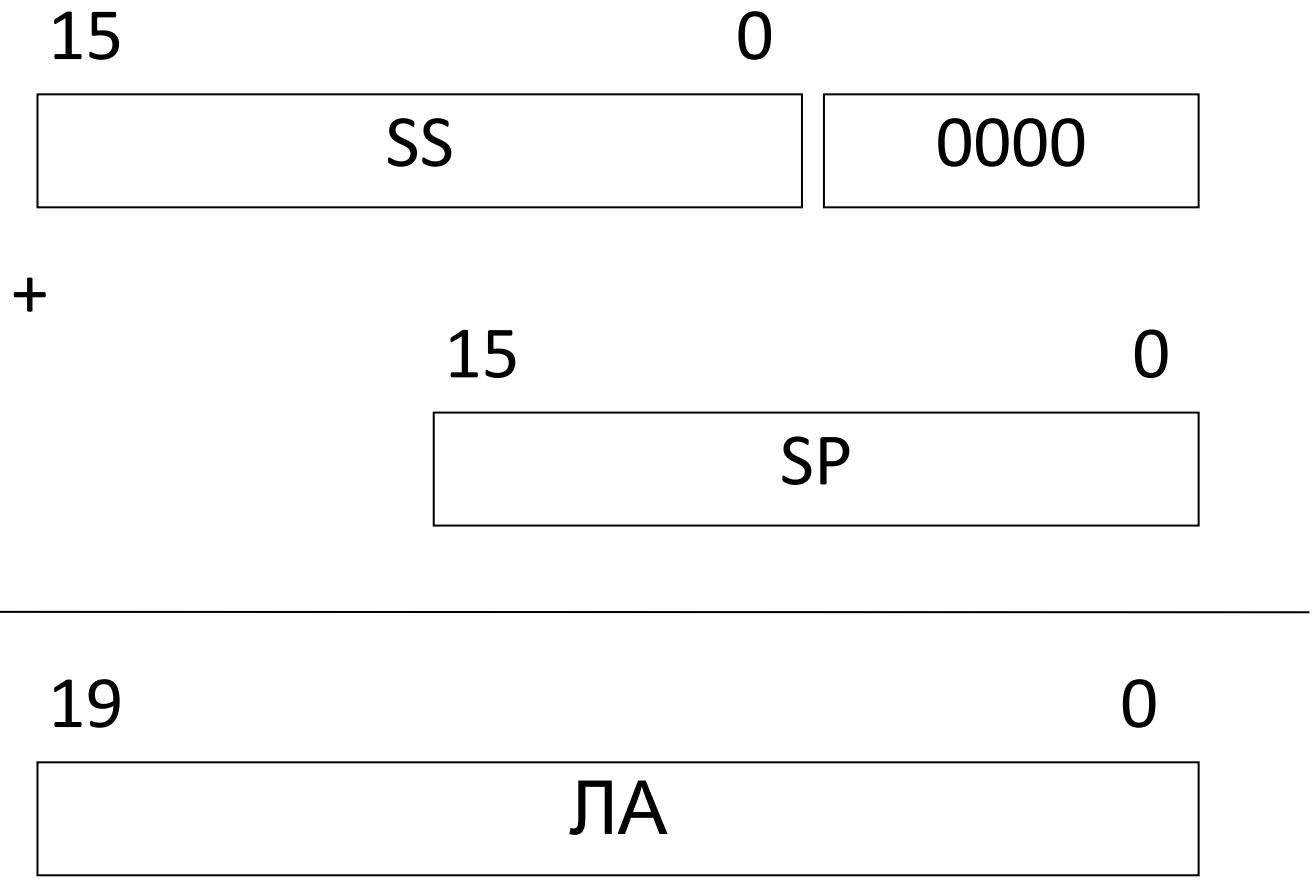
Выборка команды из памяти



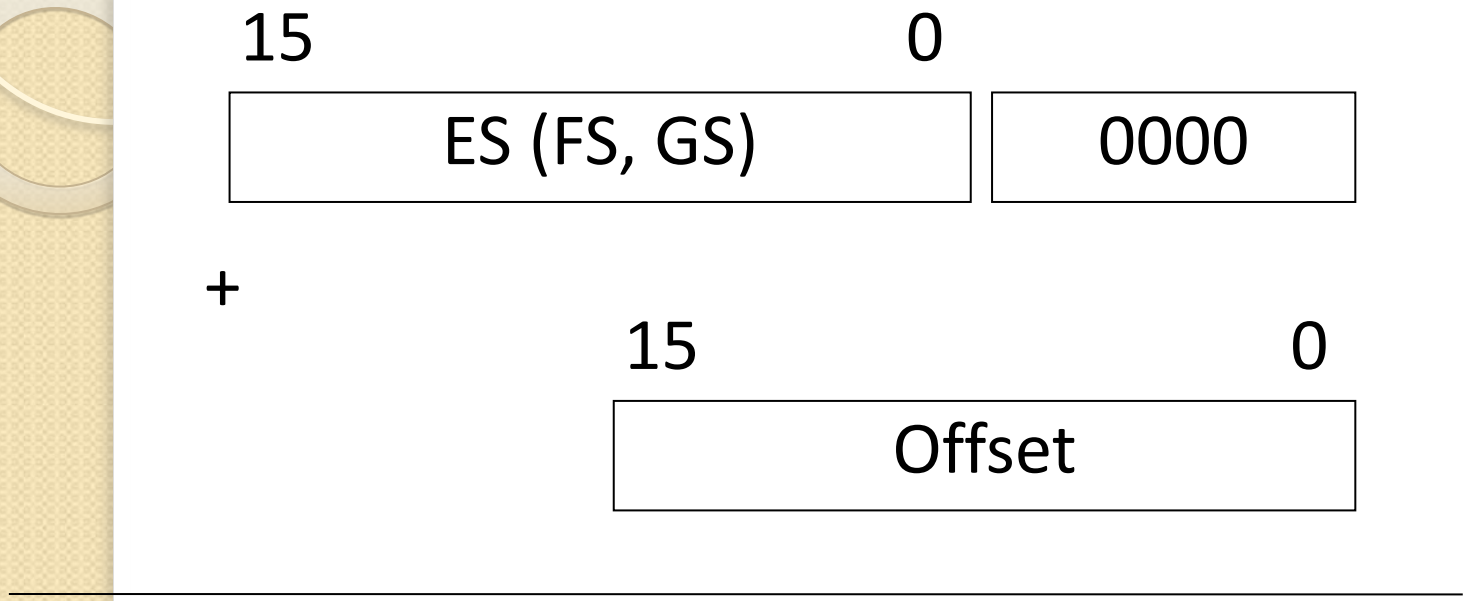
Обращение к данным



Обращение к стеку



Обращение к доп. сегментам



Пример

Адрес начала сегмента данных DS=2320h. В начале сегмента данных расположены 2 переменные: A (2 байта) и B (4 байта).

Найти:

- ЛА переменной C[], расположенной в данном сегменте следом за этими переменными и представить его в виде DS:Offset.
- Если непосредственно за сегментом данных расположен сегмент кода, то каким может быть максимальный размер переменной C[], чтобы не выйти за пределы сегмента данных в случаях:

  CS=2400h;

  CS=3400h.
- Каково будет смещение при попытке обращения к элементу массива C[] с номером 65522 , предполагая что каждый эл-т занимает 1 байт.

Хранение данных

При хранении данных в памяти младшие байты хранятся по младшему адресу, а старшие – по старшему (**little endian**).

Пример. Число 0401h длина 2 байта. Адрес 4806h. Содержимое памяти:

	01	04	
4805h	4806h	4807h	4808h

Это свойство полностью автоматизировано, поэтому при загрузке слова по адресу 4806h в регистр AX, он будет иметь вид:

	AH	AL
AX	04	01

Режимы адресации

Понятие режима адресации

Режим адресации памяти – это схема преобразования адресной информации об операнде ассемблерной команды в его исполнительный адрес (EA).

Прямая адресация – адресация, когда адрес операнда содержится непосредственно в команде.

Косвенная адресация – в команде указан регистр, в котором содержится адрес операнда.

Относительная адресация – адресация осуществляется относительно некоторого сегментного регистра.

Ограничение – в памяти может находиться не более одного операнда команды.

1. Регистровая прямая адресация

- Операнд находится в одном из регистров.
- Примеры:

```
mov AL, DH
mov BX, SI
add DX, CX
```

2. НЕПОСРЕДСТВЕННАЯ АДРЕСАЦИЯ

- В команде указывается непосредственное значение одного из операндов.
- Примеры:

```
mov AX, 0F235h
mov BL, 25h
add BX, 25h
```

3. Прямая адресация

- Адрес операнда задан непосредственно в команде.
- Пример:

```
; Сегмент данных
mem0 dw 1; Слово памяти содержит 1
mem1 dw 0; Слово памяти содержит 0
mem2 db 230; Байт памяти содержит 230
. . .
; Сегмент команд
inc mem1 ; Содержимое слова mem1  увеличивается на 1
mov DX, mem1 ; Содержимое слова mem1 заносится в DX
mov AL, mem2 ; Содержимое байта mem2 заносится в AL
```

- Конструкция offset позволяет использовать в команде не значение, а смещение какой-либо переменной.
- Пример:

```
mov DX, offset mem1 ; Смещение слова mem1 заносится в DX
```

3. Прямая адресация (продолжение)

- Адрес переменной состоит не только из смещения, но и адреса сегмента.
- Адрес сегмента берется из сегментного регистра, связанного командой assume с соответствующим сегментом (по умолчанию - DS):
  Пример:

```
assume ES: ИмяСегментаДанных
```

- Можно явно указать сегментный регистр в команде.
  Пример:

```
mov AX,  ES:0 ; Занести в AX слово со смещением 0 из
                ; сегмента, адресуемого регистром ES
```

4. Косвенная регистровая адресация

- В команде указывается регистр, содержащий смещение операнда
- В МП8086 для косвенной адресации допускается использовать только регистры BX, BP, SI, DI.
- При использовании регистров BX, SI, DI по умолчанию (если не задано явно) используется сегмент, адресуемый регистром DS.
- При использовании BP – сегмент, адресуемый регистром SS.

Пример:

```
mov [BX], AX ; Содержимое AX заносится по адресу Seg:Off,
    ; где Seg– содержимое DS, а Off– содержимое BX
mov ES:[BX], AL ; Содержимое AL заносится по адресу Seg:Off, где
                    ; Seg– содержимое ES, а Off– содержимое BX ♦
mov BX, [AX] ; Ошибка! AX не используется при косвенной
    адресации
```

5. Базовый режим адресации

- Исполнительный адрес (смещение) равен сумме одного из базовых регистров (BX, BP) и смещения-константы.

Пример:

```
mov AX, [BX]10 ; в AX заносится слово, находящееся по EA,
                 ; указанному в BX и увеличенному на 10.
```

- Допускаются разные формы записи для этого режима.

Пример:

```
mov AX, [BX+10]
mov AX, [BX]+10
mov AX, 10 [BX]
```

6. Индексный режим адресации

- Исполнительный адрес операнда вычисляется как сумма содержимого одного из индексных регистров SI или DI и смещения в виде константы-адреса памяти.

Пример:

mov AX, TEMP[SI] ; в AX заносится слово, находящееся по EA:
; TEMP + содержимое SI

7. БАЗОВО-ИНДЕКСНАЯ АДРЕСАЦИЯ

Микропроцессор 8086:

База + индекс + смещение

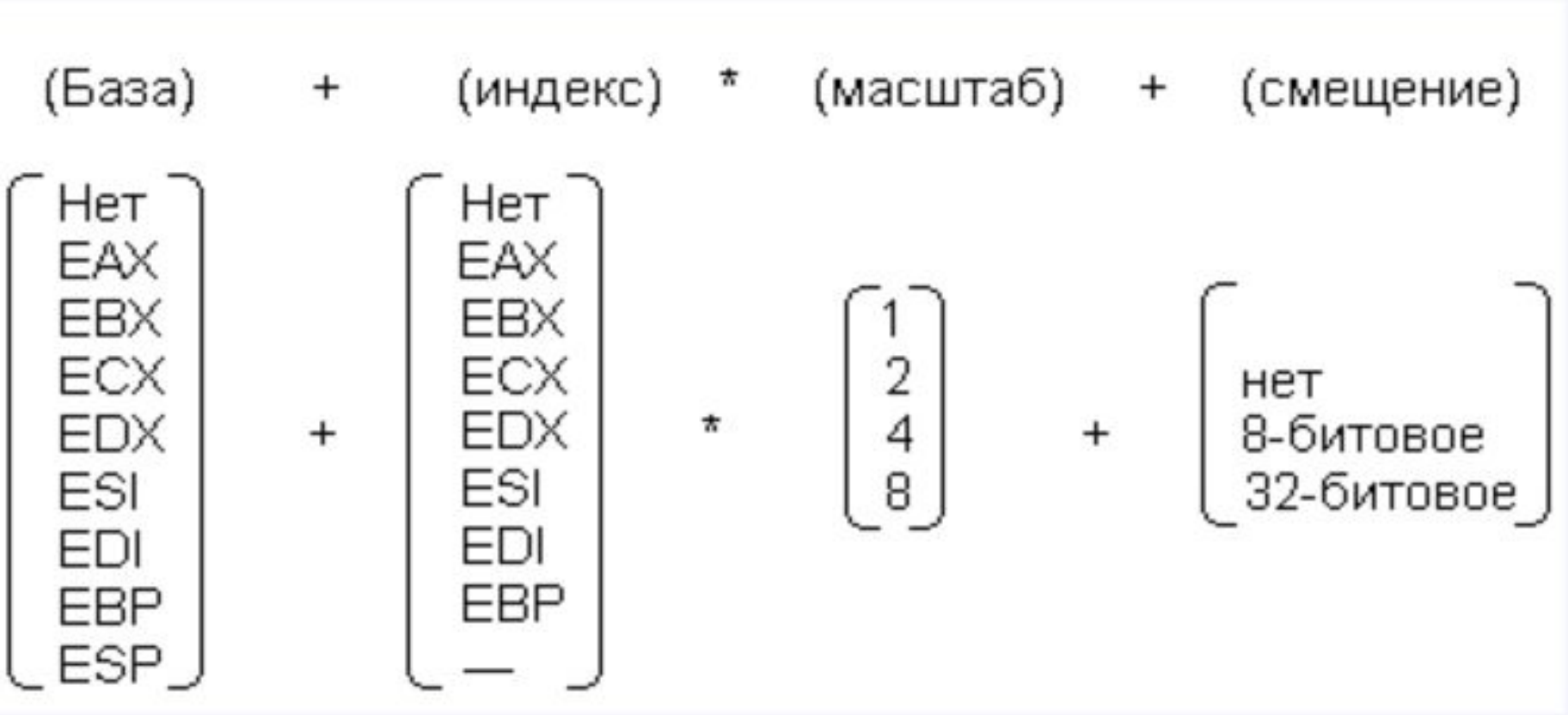
$$\begin{Bmatrix} [BX] \\ [BP] \end{Bmatrix} + \begin{Bmatrix} [DI] \\ [SI] \end{Bmatrix} + \begin{Bmatrix} disp8 \\ disp16 \end{Bmatrix}$$

Пример.

mov AX, [BX][SI]10

mov AX, TEMP[BP][DI]

7. Базово-индексная адресация МП i386



Эта схема применяется только при использовании 32-разрядных регистров!

Пример.

mov EAX, [ECX][EDI*8]10

СЕГМЕНТЫ ПРОГРАММЫ

- Сегмент кода (единственный обязательный).
- Сегмент данных.
- Сегмент стека.
- Дополнительные сегменты данных.

ОПИСАНИЕ СЕГМЕНТА

Имя	SEGMENT	<Выравнивание>	<Объединение>	‘Класс’
		Byte	Public	‘CODE’ (по ум.)
		Word	Common	‘DATA’
		Para (по умолч.)	Memory	‘CONST’
		Page	Stack	‘BSS’
			At адрес	‘STACK’
			None (по умолч.)	

. . .

Имя Ends

ПРИМЕР СТРУКТУРЫ ПРОГРАММЫ

```
STACKSG    Segment  Para   Stack ‘STACK’
        dw 80 dup (?)
STACKSG  ENDS

DATASG      Segment  Para   Public‘DATA’
. . .
DATASG  ENDS
. . .
CODESG      Segment  Para   Public‘CODE’
. . .
        assume DS:DATASG, SS:STACKSG,
ES:NOTHING
    Begin:
. . .
CODESG  ENDS
END Begin
```

ДИРЕКТИВА ASSUME

Директива ASSUME устанавливает, какой сегментный регистр используется для доступа к именам и меткам описанного ранее сегмента.

ASSUME СегмРегистр : ИмяСегмента

или

ASSUME СегмРегистр : NOTHING – отменить назначение.

Пример.

```
DATASG          Segment     Para     Public   ‘DATA’
        a db ?
DATASG ENDS
CODESG          Segment     Para     Public   ‘CODE’
begin:
        assume DS, DATASG
        mov AL, a               ; генерируется DS:a
        assume DS, NOTHING
        mov AL, DS:a            ; необходимо явно указать сегм. регистр
;
. . .
CODESG ENDS
END Begin
```

МОДЕЛИ ПАМЯТИ

Модель	*Кол-во и размер сегментов*		*Тип указателя*	
	Code	*Data*	*Code*	*Data*
MS DOS, Win 16				
Tiny	Один <=64К		Near	
Small	Один <=64К	Один <=64К	Near	Near
Medium	Несколько <=64К	Один <=64К	Far	Near
Compact	Один <=64К	Несколько <=64К	Near	Far
Large	Несколько <=64К	Несколько <=64К	Far	Far
Huge	Несколько >64К	Несколько >64К	Huge	Huge
Win 32 (начиная с i386)				
Flat	Вся память	Вся память	Flat	Flat

ДИРЕКТИВА MODEL

Задается для использования определенной модели памяти в программе:

MODEL ИмяМодели[, Язык]

Язык – позволяет упростить вопросы стыковки программ на ассемблере и на языке программирования высокого уровня. Возможные значения: C, CPP, PASCAL и др. Если язык не задан, подразумевается NOLANGUAGE.

Если в программе задана модель памяти, можно использовать упрощенные директивы описания **основных** сегментов:

CODESEG –сегмент кода;

DATASEG – сегмент данных;

STACK – сегмент стека.

ПРОГРАММЫ

```
model   SMALL
stack   100h
dataseg
        . . .    ;данные
codeseg
START:
        startupcode
        . . .    ;код
QUIT:   exitcode  0
end START
```

ПРОГРАММЫ

Загрузчик DOS устанавливает правильные адреса сегмента стека в регистре SS и сегмента кода в регистре CS. Регистр DS нужно инициализировать самостоятельно:

```
begin:   mov AX, DATASG ; в DS значение нельзя заносить
                         ; напрямую
         mov DS, AX      ; а через РОН можно
```

ЗАВЕРШЕНИЕ ПРОГРАММЫ

Завершение работы программы выполняется путем следующего вызова прерывания с номером 21h:

```
quit:   mov AL, 0       ; установка кода завершения,
                        ; передаваемого DOS
                        ; 0 – все нормально
        mov AH, 4Ch     ; код функции завершения
                        ; программы и возврата
                        ; управления DOS
        Int 21h         ; вызов прерывания
```

ОПИСАНИЕ ДАННЫХ

Вид выражения	*Пример*
Константа	c1 db 023h – в 16-ричной системе счисл. c2 db 034q – в 8-ричной системе счисл. c3 db 101b – в двоичной системе c4 db 17 – в десятичной системе
Знак вопроса – отсутствие значения	ef1 dw ?
Несколько констант, разделенных запятыми	ef2 db 11, 14, 25, 17 ef2+0 ef2+1 ef2+2 ef2+3
Повторитель dup	fl1 db 10 dup (?) ; 10 байт без значений fl2 dw 5 dup (14) ; ↔ 14,14,14,14,14
Символьную константу	fl3 db ‘+’
Символьную строку	fl4 db ‘abcde’ ; 5 байт с кодами символов
Последовательная комбинация нескольких предыдущих видов	fl5 db ‘abcde’,10 dup (0),0Ah,’!’

ДИРЕКТИВА EQU

- Не определяет никаких данных.
- Задает некоторое именованное значение, которое можно использовать в других командах.
- Имя, определенное директивой EQU, нельзя переопределить.

Пример.

```
Counter EQU 10
. . .
F1 dw Counter dup (0)
. . .
    move AX, Counter
```

ПОЛЕЗНЫЕ ДИРЕКТИВЫ ПРЕОБРАЗОВАНИЙ

PTR – однократное преобразование типа.

- Используется сатрибутами типов: BYTE, WORD, DWORD, NEAR, FAR.
- Преобразование имеет вид:

тип PTR выражение

Пример.

```
A   db  28h
    db  30h
B   dw  3344h
. . .
    mov    AL, byte ptr B      ; AL ← 44h
    mov    BX, word ptr A      ; BX ← 3028h
    mov byte ptr B, 5h    ; B ← 5h
    mov byte ptr B+1, 6h     ; B+1 ← 6h
```

ПОЛЕЗНЫЕ ДИРЕКТИВЫ ПРЕОБРАЗОВАНИЙ

OFFSET – возвращает относительный адрес переменной или метки внутри сегмента.

```
mov    AX, offset A   ; эквивалентно lea AX, A
```

SEG – возвращает адрес сегмента, в котором расположена переменная или метка.

```
mov    AX, seg A      ; AX ← DS
mov    AX, seg Label1 ; AX ← CS
```

SHORT – модификация операнда в команде перехода для сокращения кода (ускорения) машинной операции. Расстояние от команды до метки должно быть от -128 до 127 байт.

```
    jmp    short A20
. . .
A20: . . .
```

Команды ассемблера-1

Команды пересылки данных

- Команды пересылки данных позволяют переслать (скопировать) содержимое источника (<source>, <src>) в приемник (<destination>, <dst>).
- После выполнения операции пересылки содержимое приемника безвозвратно теряется, содержимое источника не изменяется.
- При выполнении команд пересылки флаги не модифицируются

Команды пересылки данных

mov dst, src – копирование данных из src в dst

dst – reg/mem, src – reg/mem/const.

Ограничения команды:

- Оба операнда должны иметь одинаковую длину.
- Пересылка типа "память-память" не поддерживается.
- В качестве получателя нельзя указывать регистры CS, и IP.
- Нельзя переслать непосредственно заданное значение в сегментный регистр.

Примеры.

mov AX, GAMMA

mov GAMMA, 05h

mov AX, [BX]

mov [DI], GAMMA – ошибка!

mov [BX], 9 – осторожно!

Команды пересылки данных

xchg op1, op2 – обмен содержимого op1 и op2

- Операнды – reg/mem.
- Действуют такие же ограничения, как и для mov.

Примеры.

xchg AX, [DI]

xchg CX, GAMMA

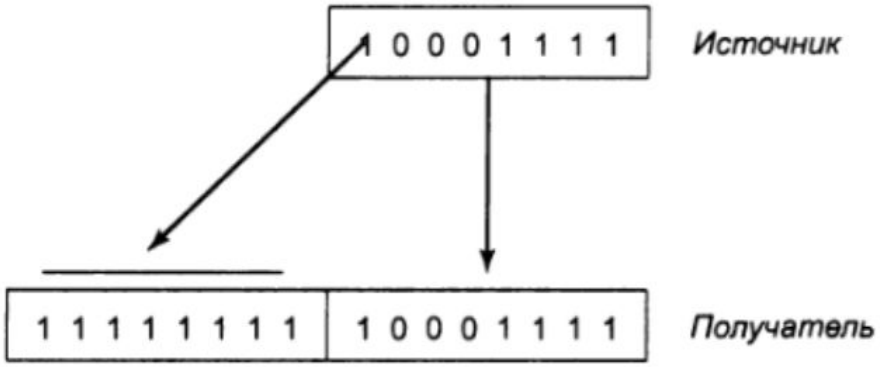
xchg AX, CL - ошибка!

Копирование со знаковым расширением

movsx dst, src – копирование со знаковым расширением (386+).

- Разрядность dst больше, чем разрядность src.
- dst – регистр, src –ячейка памяти.

Разрядность dst	Разрядность src
16	8
32	8, 16
64	8, 16, 32



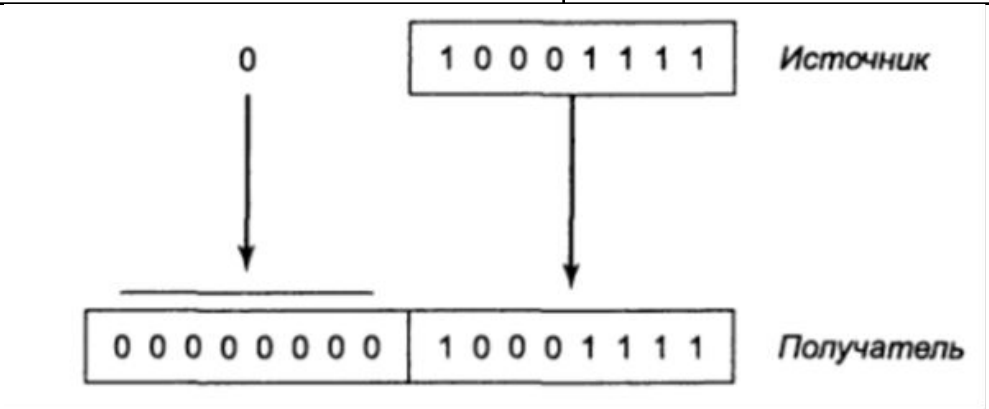
Примеры.
movsx AX, CL
movsx AX, byte ptr mem1

Копирование с нулевым расширением

movsz dst, src – копирование с нулевым расширением (386+) .

- Разрядность dst больше, чем разрядность src.
- dst – регистр, src –ячейка памяти.

Разрядность dst	Разрядность src
16	8
32	8, 16
64	8, 16, 32



Примеры.
movsz AX, CL
movsz AX, byte ptr mem1

Команды загрузки адреса данных

lea reg, mem – загрузка адреса данных (смещения)

Пример. Две аналогичные команды.

```
lea BX, GAMMA
mov BX, offset GAMMA
```

lds reg, mem – одновременная загрузка смещения в reg и сегментного адреса в DS.

- Размер mem – двойное слово (для 16-разрядных РОН) и 6 байт (48 бит) для 32-разрядных.
- В памяти сначала находится смещение, а затем – сегментный адрес.

Пример.

```
lds BX, mem ; BX←mem, DS ←mem+2
```

аналогична командам:

```
mov BX, word ptr mem
mov AX. Word ptr mem+2
mov DS, AX
```

Команды загрузки адреса данных

les reg, mem

lfs reg, mem (386+)

lgs reg, mem (386+)

lss reg, mem (386+)

Одновременная загрузка из ячейки памяти смещения в reg и сегментного адреса в указанный командой сегментный регистр (ES, FS, GS или SS). Команды аналогичны lds.

Команды пересылки флагов

LAHF - загрузка в регистр AH младшего байта регистра флагов.

- В AH копируются флаги : SF (знак), ZF (нуль), AF (служебный перенос), PF (четность) и CF (переноса).
- Флаги обычно сохраняется для дальнейшего анализа:

Пример.

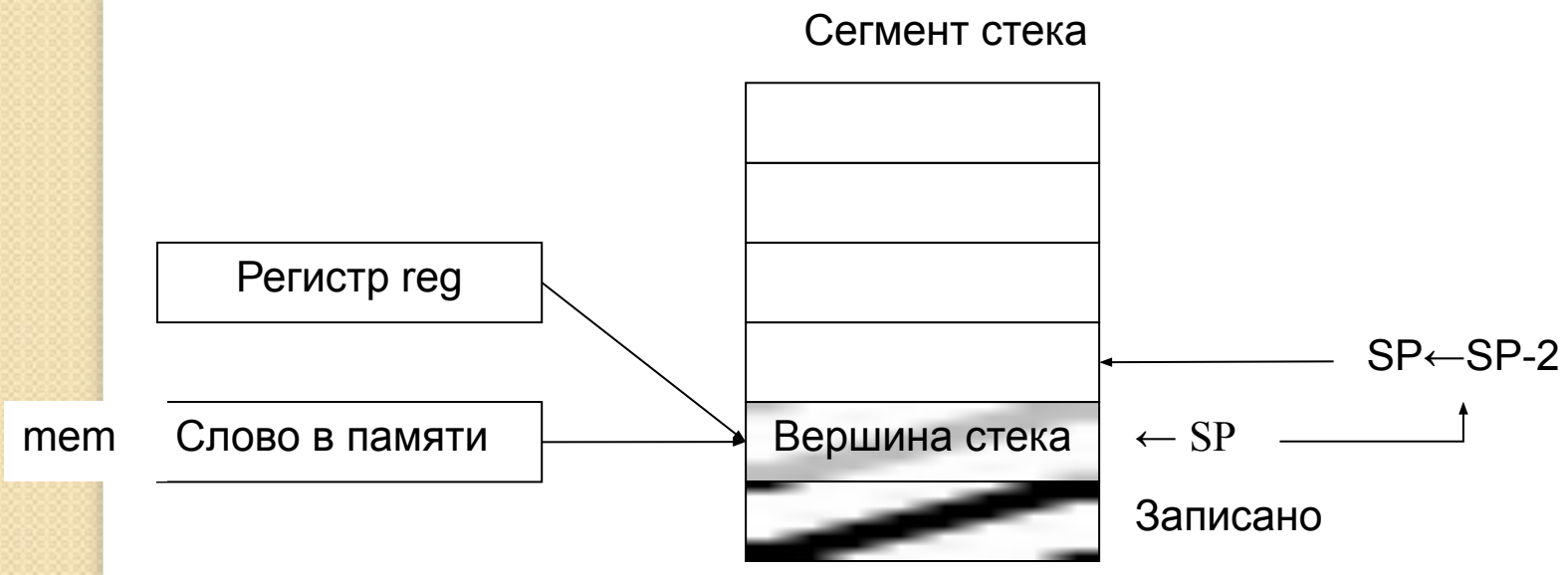
```
LAHF
MOV mem1,AH
```

SAHF - загрузка из регистра AH в младший байт регистра флагов. Команда, обратная LAHF

Команда записи в стек

push mem/reg - записать в стек

- Указатель стека SP по умолчанию автоматически уменьшается на 2.
- Слово, адресуемое указателем стека SP, называется вершиной стека.



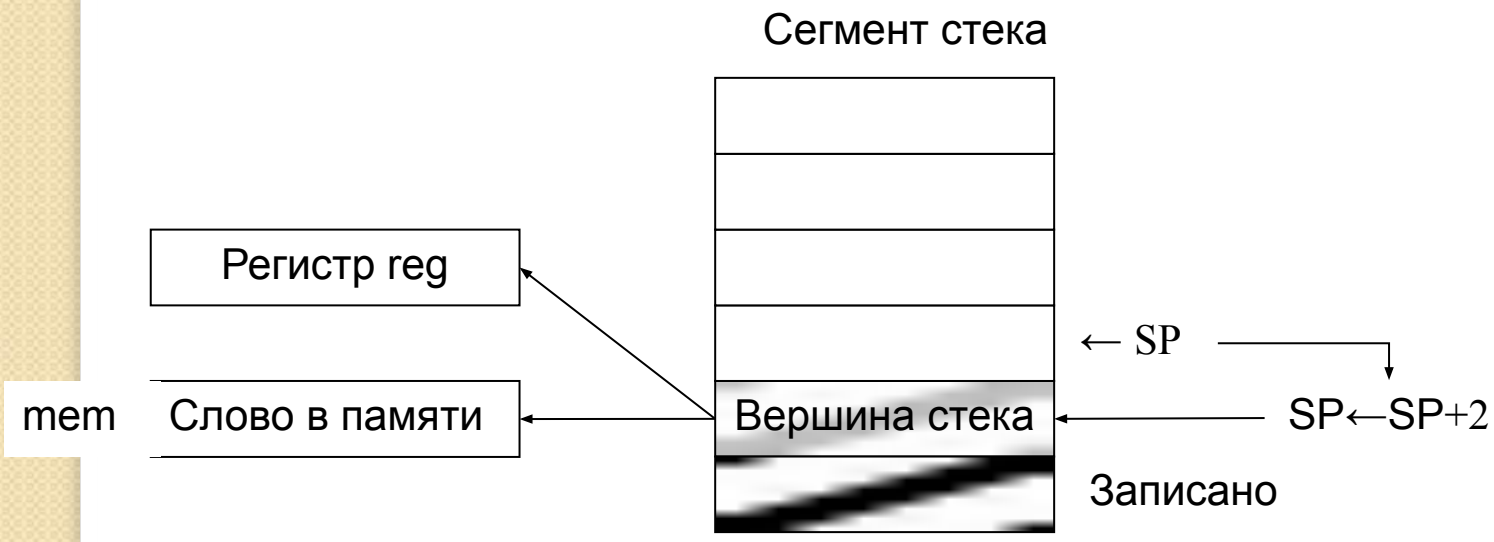
Пример.

```
push AX        ; запись AX в вершину стека
push GAMMA     ; запись слова по адресу GAMMA в вершину стека
```

Команда извлечения из стека

pop mem/reg - вытолкнуть значение из стека и записать его в ячейку памяти или регистр

- Указатель стека SP по умолчанию автоматически увеличивается на 2.
- Вершиной стека становится освобожденная ячейка.



386+. Допускается использование 32-разрядных регистров и ячеек памяти.

```
push EAX
pop EAX
push dword ptr mem
pop dword ptr mem
```

Занесение в стек и извлечение из стека регистра флагов

pushf

Занесение в вершину стека 16-битного регистра флагов. SP←SP-2.

popf

Выталкивает и заносит в 16-битный регистр флагов сохраненное ране значение. SP←SP + 2.

386+

pushfd – сохранение в стеке 32-разрядного регистра флагов. SP←SP-4.

popfd – извлечение из стека 32-разрядного регистра флагов. SP←SP+4.

Занесение в стек и извлечение из стека всех РОН

pusha

Запись в стек сразу всех 8 16-разрядных РОН в таком порядке: AX, CX, DX, BX, SP, BP, SI, DI. SP←SP-16.

popa

Извлечение из стека и занесение в РОН всех 8 16-разрядных регистров. Значение регистра SP, сохраненное в стеке, командой извлекается, но отбрасывается. SP←SP+16.

386+

pushad

Запись в стек сразу всех 8 32-разрядных РОН в таком порядке: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. SP←SP-32.

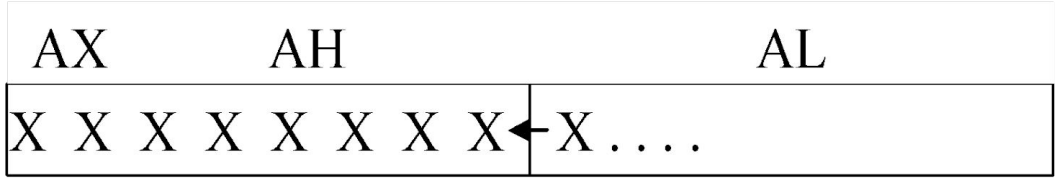
popad

Извлечение из стека и занесение в РОН всех 8 16-разрядных регистров. Значение регистра SP, сохраненное в стеке, командой извлекается, но отбрасывается. SP←SP+32.

Расширение разрядности знаковых чисел в регистрах

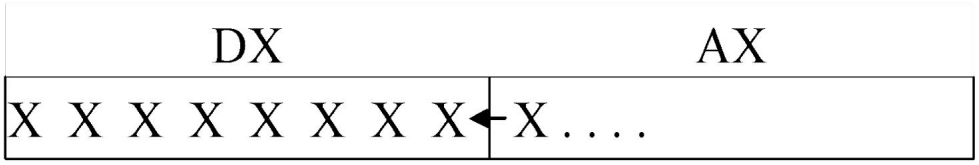
cbw

- AX расширяется из AL.
- AH заполняется старшим разрядом регистра AL.
- После команды AX = AL (только для знаковых чисел).



cwd

- DX заполняется знаковым разрядом из AX.
- После заполнения пара регистров DX, AX содержит 32-разрядное число, равное числу в AX (только для знаковых чисел).



Расширение разрядности знаковых чисел в 32-разрядных регистрах (386+)

cwde

- EAX заполняется знаковым разрядом из AX.
- После расширения EAX=AX (только для знаковых чисел).

cdq

- EDX заполняется знаковым регистром из EAX.
- После пара регистров EDX, EAX содержит 64-разрядное знаковое число, равное знаковому числу в EAX.

Команды ассемблера - 2

Команды двоичной арифметики

- Предназначены для выполнения базовых арифметических операций
- По результатам выполнения устанавливаются/сбрасываются флаги
- После выполнения можно проверить флаги и принять решение о выполнении той или иной ветки алгоритма
- Операнды:
  - dst – mem/reg
  - src – mem/reg/data
- Разрядности операндов должны совпадать

Команды сложения

add dst, src ; dst = dst + src

adc dst, src ; dst = dst + src + CF (сложение с переносом)

Команда adc используется при реализации сложения чисел удвоенной разрядности.

Пример. Сложение 2 32-разрядных чисел с использованием 16-разрядных регистров

```
        dataseg
num1 dd 1252349
num2 dd 3246728
num3 dd ?
        codeseg
begin:
…
        mov AX, num1
        mov BX, num1+2
        mov CX, num2
        mov DX, num2+2
        add AX, CX
        adc BX, DX
        mov num3, AX
        mov num3+2, BX
…
end begin
```

Команды вычитания

sub dst, src ; dst = dst – src

sbb dst, src ; dst = dst – src - CF (вычитание с заемом)

Команда **sbb** используется при реализации вычитания чисел удвоенной разрядности.

Пример. Вычитание 2 32-разрядных чисел с использованием 16-разрядных регистров

```
        dataseg
num1 dd 1252349
num2 dd 3246728
num3 dd ?
        codeseg
begin:
        mov AX, num1
        mov BX, num1+2
        mov CX, num2
        mov DX, num2+2
        sub AX, CX
        sbb BX, DX
        mov num3, AX
        mov num3+2, BX
end begin
```

Сложение с обменом

xadd mem, reg ; 486+

Выполняется сложение операнда из памяти с содержимым регистра. После производится обмен: содержимое ячейки памяти заносится в регистр, а результат сложения – в ячейку памяти.

При выполнении команд сложения и вычитания возникает вопрос: как определить, что результат операции вышел (не вышел) за границы области возможных значений?

Правило.

- Для беззнаковых чисел признаком выхода результата за границу диапазона является единица во флаге CF (CF=1).
- Для знаковых чисел – единица во флаге OF (OF=1).

Дополнительные арифметические команды

inc dst ; dst = dst + 1

dec dst ; dst = dst – l

neg dst ; dst = - dst

cmp op1, op2 ; op1 – op2 (без сохранение результата, влияет только на регистр флагов).

Действие команд двоичной арифметики на флаги

	OF	SF	ZF	AF	PF	CF
add	+	+	+	+	+	+
adc	+	+	+	+	+	+
sub	+	+	+	+	+	+
sbb	+	+	+	+	+	+
xadd	+	+	+	+	+	+
inc	+	+	+	+	+	–
dec	+	+	+	+	+	–
neg	+	+	+	+	+	+
cmp	+	+	+	+	+	+

Команды побитовых логических операций

- Побитовые логические операции рассматривают операнды как последовательность бит.
- Операция выполняется между каждой парой соответствующих бит.
- Ограничения этих команд такие же, как и для команды MOV.
- Dst – ячейка памяти или регистр, Src – ячейка памяти, регистр или непосредственное значение.

Команды побитовых логических операций

notdst – побитовая инверсия.

Каждый бит dst меняет свое значение на противоположное.

or dst, src – логическое «ИЛИ».

Каждый бит dst вычисляется по правилам операции OR:

dst = dst OR src.

and dst, src – логическое «И».

Каждый бит dst вычисляется по правилам операции AND:

ёdst = dst AND src.

xordst, srx – «ИСКЛЮЧАЮЩЕЕ ИЛИ».

Каждый бит dst вычисляется по правилам операции XOR: dst = dst XOR src.

test op1, op2 - логическое «И» без сохранение результата (влияет только на регистр флагов).

Правила выполнения побитовых операций

Dst	Src	dst			
		Not	And	Or	Xor
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

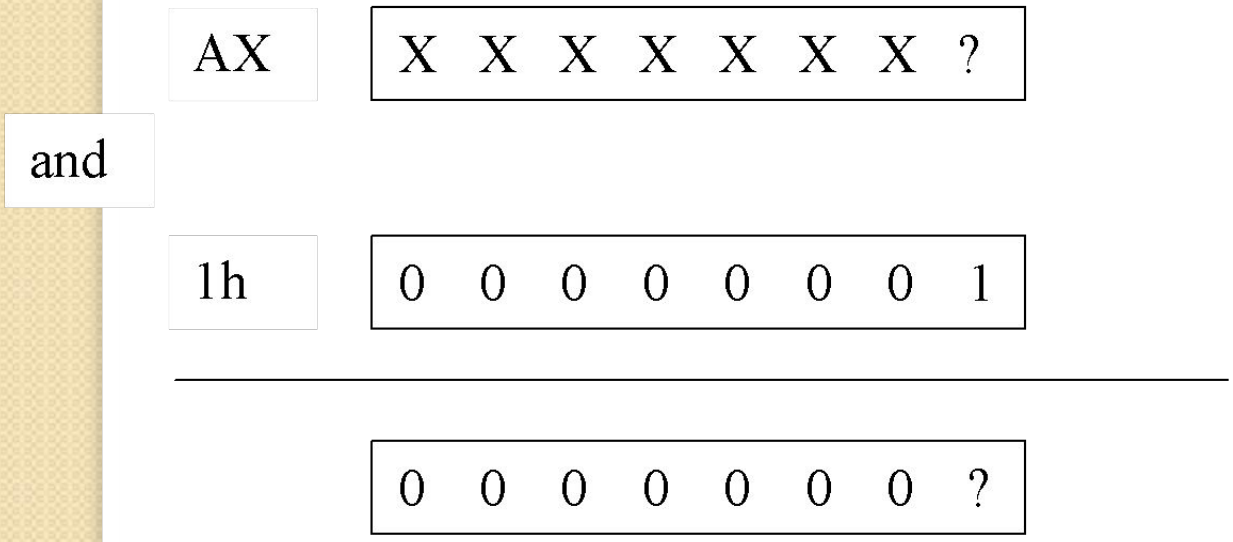
Действие команд на флаги

	OF	SF	ZF	AF	PF	CF
Not	–	–	–	–	–	–
Остальные	–	+	+	Не опр	+	–

Примеры использования побитовых логических команд

Пример проверки бита. Проверить, является ли младший бит регистра AX единицей.

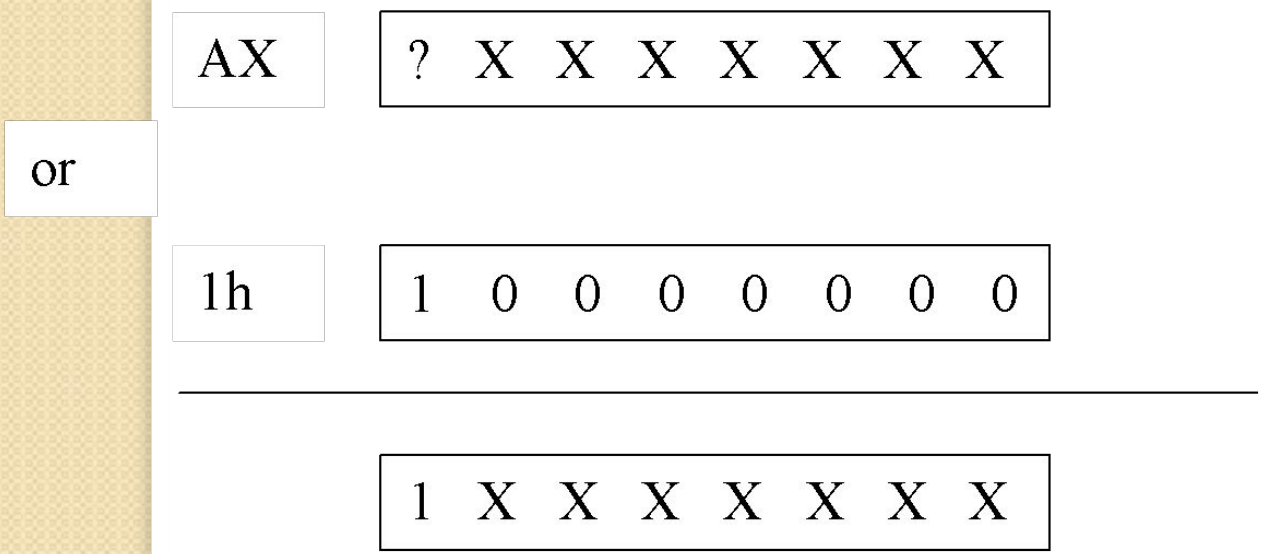
test AX, 1h ; если после этой команды флаг ZF=0, то является,
; иначе– 0



Примеры использования побитовых логических команд

Пример установки бита. Установить в регистре AX старший бит в единицу.

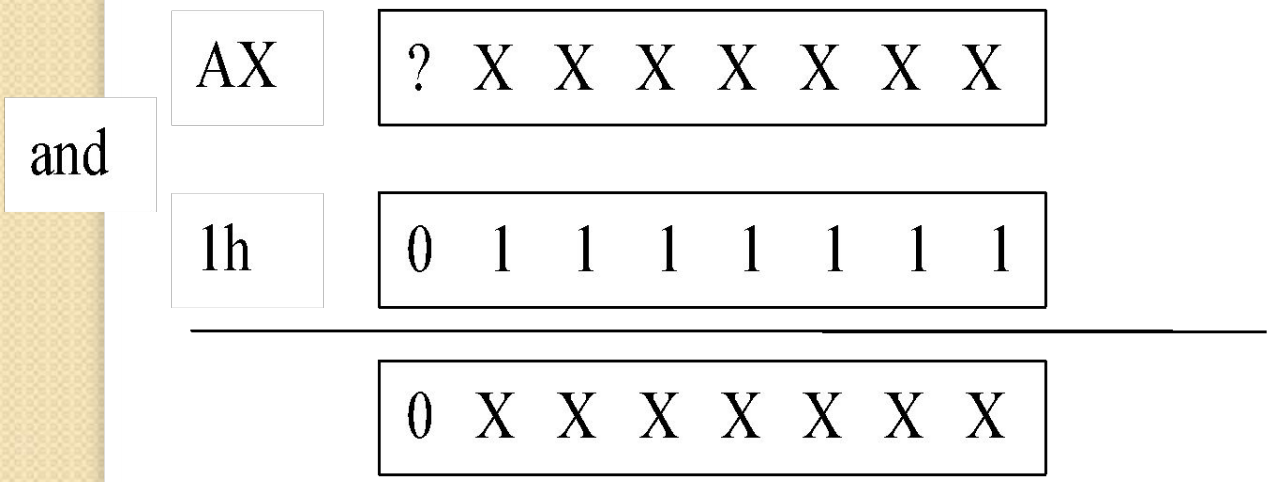
or AX,8000h



Примеры использования побитовых логических команд

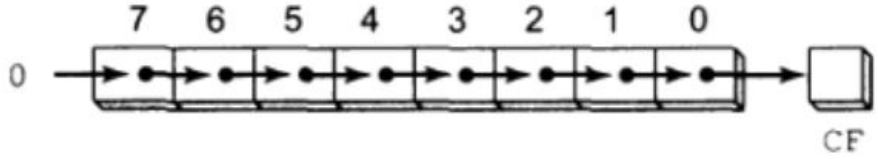
Пример обнуления бита. Обнулить в регистре AX старший бит.

and AX, 7FFFh

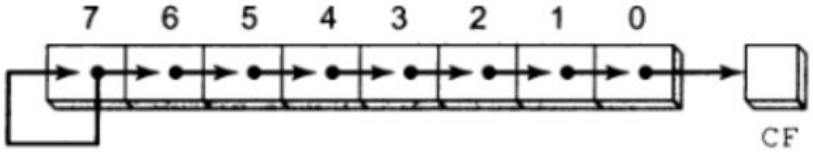


Операции сдвига

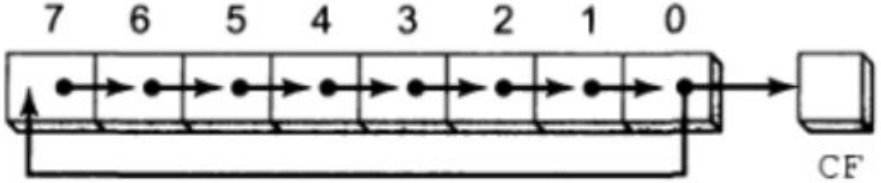
- **Логический сдвиг**- освобождающиеся разряды заполняются нулями.



- **Арифметический сдвиг**. Во время его выполнения освобождающиеся разряды заполняется первоначальным значением знакового разряд



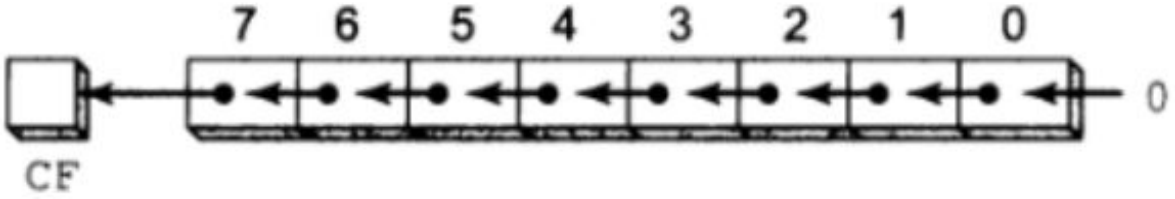
- **Циклический сдвиг**. При его выполнении разряды, перемещаемые за разрядную сетку, помещаются в освобождаемые разряды.



Команды логического сдвига

shl dst, count - логический сдвиг влево.

- dst – ячейка памяти или регистр;
- count – регистр CL или число, в обеих случаях значение должно быть в диапазоне от 1 до 7/15/31/63.
- Выполняет логический сдвиг влево операнда dst данных на количество разрядов, указанных в count.
- Младшие (освобождающиеся) разряды заполняются нулями.
- Старшие разряды числа последовательно помещаются во флаг переноса CF, а бит, который до этого находился во флаге переноса, теряется.

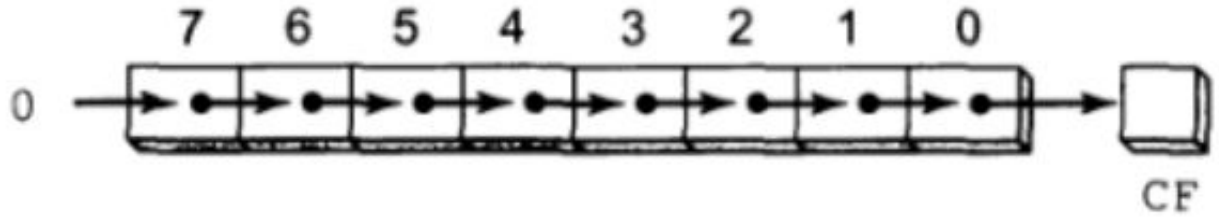


shl dst, count
эквивалентен по результату применению count команд
shl dst, 1

Команды логического сдвига

shr dst, count - логический сдвиг вправо

- dst – ячейка памяти или регистр;
- count – регистр CL или число, в обеих случаях значение должно быть в диапазоне от 1 до 7/15/31/63.
- Выполняет логический сдвиг вправо операнда dst на количество разрядов, указанных в count.
- Старшие (освобождающиеся) разряды заполняются нулями.
- Младшие разряды числа последовательно помещаются во флаг переноса CF, а бит, который до этого находился во флаге переноса, теряется.



shr dst, count
эквивалентен по результату применению count команд
shr dst, 1

Применение команд логического сдвига

- Используются для работы с битовыми полями в комбинации с побитовыми логическими командами.
- Используются для реализации быстрого умножения/деления беззнаковых чисел на 2^n.

Пример работы с битовыми полями. Пусть содержимое регистра AX содержит битовые поля со значениями дня, месяца и года:

Поле	*Назначение*	*Диапазон значений*
0-4 бит	День	0 – 31
5-8 бит	Месяц	0 –15
9-15 бит	Год	0 –127 (смещение отн. 1980

Требуется выделить или установить значение каждого поля.

Применение команд логического сдвига

Выделение значения каждого из полей.

```
mov     BX, AX      ; сохраняем копию AX
and     AX, 1Fh     ; в AX остался только день
…
mov     AX, BX ; восстанавливаем исходное содержимое AX
shr     AX, 5   ; сдвигаем так, чтобы с нулевого бита
начиналось поле месяца
and     AX, 0Fh     ; в AX остался только месяц
…
mov     AX, BX ; восстанавливаем исходное содержимое AX
shr AX, 9   ; сдвигаем так, чтобы с нулевого бита начиналось
поле года
```

Применение команд логического сдвига

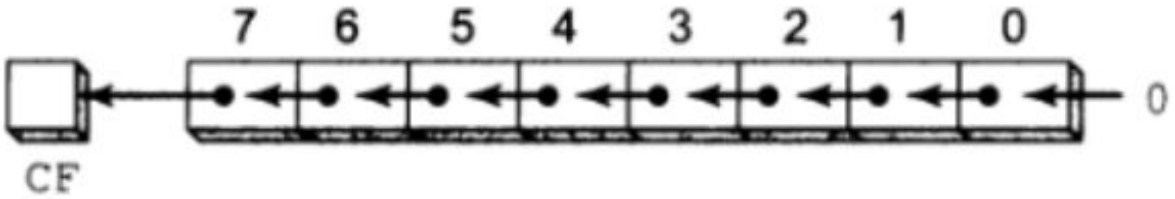
Установка значения каждого из полей в регистр AX.

```
mov    BX, Day    ; занесли день в BX
and    AX, 0FFE0h    ; обнуляем день в AX
or  AX, BX     ; установили день в AX
…
mov    BX, Month ; занесли месяц в BX
shl BX, 5      ; сдвигаем значение месяца в свои биты BX
and    AX, 0FE1Fh    ; обнуляем месяц в AX
or  AX, BX     ; установили месяц в AX
…
mov    BX, Year   ; занесли год в BX
shl BX, 9      ; сдвигаем значение года в свои биты BX
and    AX, 1FFh   ; обнуляем год в AX
or  AX, BX     ; установили год в AX
```

Команды арифметического сдвига

sal dst, count - арифметический сдвиг влево

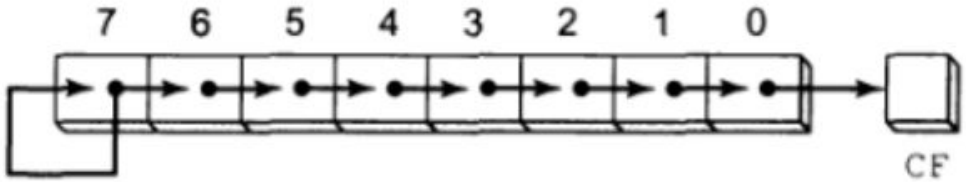
- dst – ячейка памяти или регистр;
- count – регистр CL или число, в обеих случаях значение должно быть в диапазоне от 1 до 7/15/31/63.
- Выполняет арифметический сдвиг влево операнда dst данных на количество разрядов, указанных в count.
- Младшие (освобождающиеся) разряды заполняются нулями.
- Старшие разряды числа последовательно помещаются во флаг переноса CF, а бит, который до этого находился во флаге переноса, теряется.
- Полностью аналогичен shl.



Команды арифметического сдвига

sardst, count - арифметический сдвиг вправо

- dst – ячейка памяти или регистр;
- count – регистр CL или число, в обеих случаях значение должно быть в диапазоне от 1 до 7/15/31/63.
- Выполняет арифметический сдвиг вправо операнда dst данных на количество разрядов, указанных в count.
- Старшие (освобождающиеся) разряды заполняются знаковым разрядом.
- Младшие разряды числа последовательно помещаются во флаг переноса CF, а бит, который до этого находился во флаге переноса, теряется.



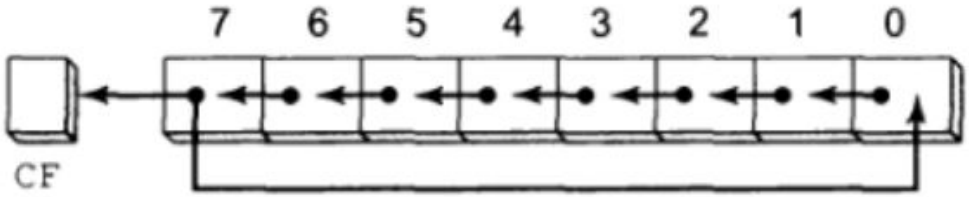
Пример. Реализовать деление числа X на 8 (с отбрасыванием остатка).

```
mov     AX, X
sar AX, 3          ; AX = X/2^3
```

Команды циклического сдвига

rol dst, count - циклический сдвиг влево

- dst – ячейка памяти или регистр;
- count – регистр CL или число, в обеих случаях значение должно быть в диапазоне от 1 до 7/15/31/63.
- Циклически сдвигает каждый бит операнда dst влево на количество разрядов, указанных в count.
- Старшие биты числа последовательно копируются в младший бит, а также во флаг переноса CF.



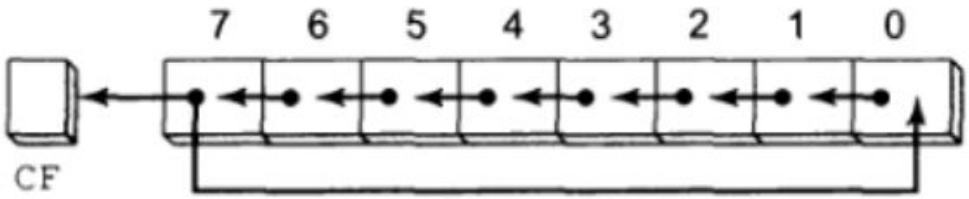
Пример. Обмен старшего (биты 4—7) и младшего (биты 0-3) полубайтов числа:

```
mov   AL, 26h
rol   AL, 4        ; AL ← 62h
```

Команды циклического сдвига

ror dst, count - циклический сдвиг вправо

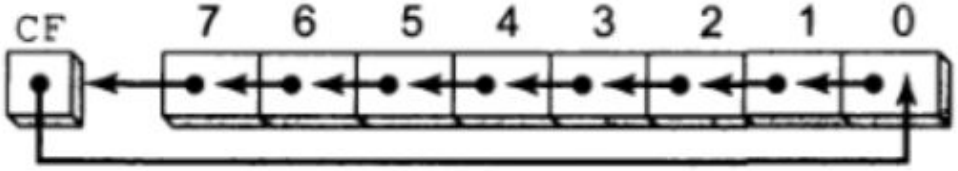
- dst – ячейка памяти или регистр;
- count – регистр CL или число, в обеих случаях значение должно быть в диапазоне от 1 до 7/15/31/63.
- Циклически сдвигает каждый бит операнда dst вправо на количество разрядов, указанных в count.
- Младшие биты числа копируются в старший бит, а также во флаг переноса CF.



Команды циклического сдвига с переносом

rcl dst, count - циклический сдвиг влево через флаг переноса

- dst – ячейка памяти или регистр;
- count – регистр CL или число, в обеих случаях значение должно быть в диапазоне от 1 до 7/15/31/63.
- Циклически сдвигает через флаг переноса каждый бит dst влево на количество разрядов, указанных в count.
- Значение флага переноса CF последовательно помещается на место самого младшего бита, а самый старший (знаковый) бит числа после



еноса CF.

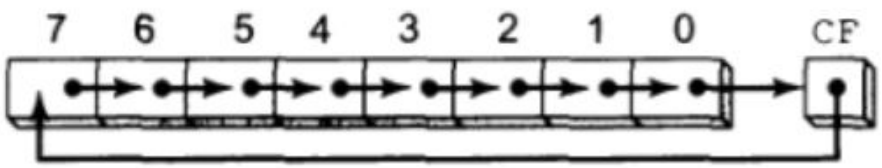
Пример. Проверить значение младшего бита регистра AX.

```
shr AX, 1    ; в CF младший бит регистра
…
rcl  AX, 1    ; значение AX восстановлено
```

Команды циклического сдвига с переносом

rcr dst, count - циклический сдвиг вправо через флаг переноса

- dst – ячейка памяти или регистр;
- count – регистр CL или число, в обеих случаях значение должно быть в диапазоне от 1 до 7/15/31/63.
- Циклически сдвигает через флаг переноса каждый бит операнда dst вправо на количество разрядов, указанных в count.
- Значение флага переноса CF последовательно помещается на место самого старшего бита, а самый младший бит копируется во флаг переноса CF



Пример. Проверить значение бита 2 (начиная с 0) регистра AX.

```
rcr  AX, 3    ; в CF бит 2 регистра AX
…
rcl  AX, 3    ; значение AX восстановлено
```

Действие команд сдвига на флаги

	OF	SF	ZF	AF	PF	CF
shl	+	+	+	Не опред.	+	+
shr	+	+	+	Не опред.	+	+
sal	+	+	+	Не опред.	+	+
sar	+	+	+	Не опред.	+	+
rol	+			Не опред.		+
ror	+			Не опред.		+
rcl	+			Не опред.		+
rcr	+			Не опред.		+

Команды сканирования битов

bsf reg, reg/mem

- Сканирует биты второго операнда, начиная с младшего бита, до тех пор, пока не будет найден бит, равный 1.
- Его номер заносится в регистр, указанный в первом операнде (биты нумеруются с 0).
- Если нет ни одного единичного бита, устанавливается флаг ZF.
- Операнды должны иметь равную длину.

Пример

```
mov AX, 050h
bsf DX, AX       : DX=4
```

bsr reg, reg/mem

- Сканирует биты второго операнда, начиная со старшего бита, до тех пор, пока не будет найден бит, равный 1.
- Его номер заносится в регистр, указанный в первом операнде (биты нумеруются с 0).
- Если нет ни одного единичного бита, устанавливается флаг ZF.
- Операнды должны иметь равную длину.

Пример

```
mov AX, 050h
bsr DX, AX       : DX=6
```

Команды сканирования битов

bt reg/mem, reg/data

- Анализирует бит, номер которого задан вторым операндом, в значении, заданном первым операндом (биты нумеруются с 0).
- Заданный бит заносится в флаг CF.
- Операнды должны иметь равную длину.

Пример.

```
Bt AX, 4
jc yes
      ; обработка если бит равен 0
jmp m1
yes:     ; обработка если бит равен 1
m1:
```

Команды сканирования битов

btc reg/mem, reg/data

- Анализирует бит, номер которого задан вторым операндом, в значении, заданном первым операндом (биты нумеруются с 0).
- Проверяемый бит заносится во флаг CF.
- Проверяемый бит инвертируется.

btr reg/mem, reg/data

- Анализирует бит, номер которого задан вторым операндом, в значении, заданном первым операндом (биты нумеруются с 0).
- Проверяемый бит заносится во флаг CF.
- Проверяемый бит обнуляется.

bts reg/mem, reg/data

- Анализирует бит, номер которого задан вторым операндом, в значении, заданном первым операндом (биты нумеруются с 0).
- Проверяемый бит заносится во флаг CF.

Команды ассемблера - 3

Команды переходов

Классификация переходов:

1. По модифицируемым регистрам.
   - NEAR – внутрисегментный, «ближний» (модифицируется только регистр IP);
   - FAR – межсегментный, «дальний» (модифицируются CS:IP)
2. По условию выполнения перехода.
   - безусловный – переход выполняется всегда;
   - условный – переход выполняется в случае, если комбинация проверяемых флагов истинна.
3. По способу задания адреса перехода.
   - Прямой – переход на заданную в программе метку.
   - Косвенный – переход по адресу, задаваемому через РОН.

Команда безусловного перехода:

jmp адрес – переход на метку/адрес

Пример.

```
jmpLabel_1    ; переход на инструкцию, помеченную меткой Label_1
jmp[BX]       ; переход на адрес, находящийся в памяти по адресу,
              ; содержащемуся в BX
```

Условный переход

Последовательность применения:

1. Использовать команду, модифицирующую флаги:

cmp op1. op2 ; op1 – op2

test op1. op2 ; op1 & op2

Любая другая команда

2. Выполнить команду условного перехода.

Команда метка

- Переход на метку, если условие команды истинно.
- Переход к следующей команде, если условие команды ложно.

Команды условных переходов

Мнемокод	Аналог	Проверяемые флаги (условие перехода)	Используется для организации перехода, если…
jz	je	ZF=1	…результат=0 …операнды равны
jnz	jne	ZF=0	…результат<>0 …операнды не равны
js	-	SF=1	…результат отрицательный
jns	-	SF=0	…результат неотрицательный
jo	-	OF=1	… переполнение
jno	-	OF=0	…нет переполнения
jp	jpe	PF=1	… в результате четное число единиц
jnp	jpo	PF=0	… в результате нечетное число единиц
jcxz	-	CX=0	… регистр CX (счетчик цикла) =0
Jc	-	CF=1	
Jnc		CF=0	

Команды условных переходов при сравнении беззнаковых чисел

Мнемокод	Аналог	Проверяемые флаги (условие перехода)	Используется для организации перехода, если…
jb	jnae, jc	CF=1	… первый операнд «ниже» второго (при вычитании был перенос)
jnb	jae, jnc	CF=0	… первые операнд «выше» или равен второму
jbe	jna	CF = 1 or ZF = 1	… первый операнд «ниже» или равен второму
jnbe	ja	CF = 0 or ZF = 0	… первый операнд «выше» второго

Команды условных переходов при сравнении знаковых чисел

Мнемокод	Аналог	Проверяемые флаги (условие перехода)	Используется для организации перехода, если…
jl	jnge	SF ⊕ OF = 1	… первый операнд меньше второго
jnl	jge	SF ⊕ OF = 0	… первые операнд больше или равен второму
jle	jng	(SF ⊕ OF) or ZF = 1	… первый операнд меньше или равен второму
jnle	jg	(SF ⊕ OF) or ZF = 1	… первый операнд больше второго

Команды переходов

Реализация аналогов условных операторов if и if-else языков высокого уровня в программе на ассемблере:

```
if (A>0) then
      { Блок 1 }
end if
```

Вариант 1:

```
       cmp AX, 0
       jg Lab_1
       jmp End_If
Lab_1:
       { Блок 1 }
End_If:   ...
```

Вариант 2:

```
       cmp AX, 0
       jle End_If
       { Блок 1 }
End_If:   ...
```

```
if (A>0) then
      { Блок 1 }
else
      { Блок 2 }
end if
```

Вариант 1:

```
       cmp AX, 0
       jg Lab_1
       { Блок 2}
       jmp End_If
Lab_1:
       { Блок 1 }
End_If:   ...
```

Вариант 2:

```
       cmp AX, 0
       jle Lab_2
       { Блок 1}
       jmp End_If
Lab_2:
       { Блок 2 }
End_If:   ...
```

Команды переходов

Проверка нескольких условий в программе на ассемблере:

```
if (A>0) and (C=0) then
      { Блок 1 }
end if
```

```
        cmp AX, 0
        jle End_If      ; A<=0 – сразу выход
        cmp CX, 0
        jne End_If      ; (A>0) and (C<>0) – выход
        { Блок 1 }
End_If:   ...
```

```
if (A>0) or (C=0) then
      { Блок 1 }
end if
```

```
        cmp AX, 0
        jg Lab_1        ; A>0 - Ок
        cmp CX, 0
        jne End_If      ; (A<=0) and (C<>0) - выход
Lab_1:   { Блок 1 }
End_If:   ...
```

Команды организации циклов

Loop метка ; команда организации цикла

- В качестве беззнакового счетчика цикла всегда используется CX.
- Цикл с пост-условием (условие проверяется в конце цикла, тело цикла всегда выполняется как минимум один раз.
- Проверка условия выхода (команда loop) эквивалентна выполнению последовательности действий:

```
CX = CX-1
jnz метка ; переход на метку, если CX<>0
```

- Невозможно организовать вложенные циклы без дополнительных действий по сохранению счетчика CX
- Команда LOOP может передать управление метке только на «расстоянии» -128.. 127 байт

Схема организации цикла

```
        mov CX, <число итераций>
Start_Loop:...
        ...
        loop Start_Loop
```

Пример. Суммирование элементов массива целых чисел.

```
datasg
intarr    dw 100h,200h,300h,400h
codeseg
      mov DI, offset intarr     ; адрес первого эл-та массива
      mov CX, 4         ; счетчик цикла – кол-во эл-тов массива
      xor AX, AX        ; обнуляем сумму – AX
L1:
      add AX, [DI]            ; прибавить к сумме очередн. Эл-т
      add DI, 2         ; прибавить к DI размер эл-та массива
      loop L1           ; повторить цикл пока CX не станет 0
```

Организация вложенных циклов

Пример 1. Сохранение счетчика в памяти.

```
dataseg
count   dw  ?
        codeseg

        mov cx, 100     ; Установить счетчик внешнего цикла
L1:
        . . .       ; Тело внешнего цикла
        mov count, CX  ; Сохранить счетчик внешнего цикла
        mov cx, 20 ; Установить счетчик внутреннего цикла
 L2:
        . . .       ; Тело внутреннего цикла
        Loop l2         ; Повторить внутренний цикл
        mov cx,count    ; Восстановить счетчик внешнего цикла
        . . .       ; Тело внешнего цикла
        loop L1             ; Повторить внешний цикл
```

Организация вложенных циклов

Пример 2. Сохранение счетчика в стеке.

```
        codeseg

        mov cx, 100     ; Установить счетчик внешнего цикла
L1:
        . . .       ; Тело внешнего цикла
        push cx
        mov cx, 20 ; Установить счетчик внутреннего цикла
 L2:
        . . .       ; Тело внутреннего цикла
        Loop l2         ; Повторить внутренний цикл
        pop cx          ; Восстановить счетчик внешнего цикла
        . . .       ; Тело внешнего цикла
        loop L1             ; Повторить внешний цикл
```

Модификации команды loop

Мнемокод	Аналог	Выполняемая последовательность действий
loopz	loope	CX = CX-1; переход, если (CX<>0 and ZF=1)
loopnz	loopne	CX = CX-1; переход, если (CX<>0 and ZF=0)

Пример 1. Проверить на равенство 2 массива.

```
datasg
M1  dw 100h,200h,300h,400h
M2   dw 100h,200h,300h,400h
codeseg
     xor DI, DI          ; смещение эл-та массива
     mov CX, 4           ; счетчик цикла – кол-во эл-тов массивов
L1:  mov AX, M1[DI]          ; элемент 1-го массива в AX
     mov BX, M2[DI]          ; элемент 2-го массива в BX
     add DI, 2               ; прибавить к DI размер эл-та массива
     cmp AX, BX          ; сравнение элементов массивов
     loopz L1            ; повторить цикл если элементы равны
     jz ravno
     . . .               ; обработка если массивы не равны
     jmp endcmp
ravno:   . . .           ; обработка если массивы равны
endcmp: . . .                ; продолжение программы
```

Модификации команды loop

Пример 2. Проверить есть ли в массиве заданное число

```
datasg
M    dw 100h,200h,300h,400h
N    dw 300h
     codeseg
     xor DI, DI              ; смещение эл-та массива
     mov CX, 4          ; счетчик цикла – кол-во эл-тов массивов
L1: mov AX, M1[DI]           ; элемент массива в AX
     add DI, 2               ; прибавить к DI размер эл-та массива
     cmp AX, N          ; сравнение элемента массива и числа
     loopnz L1          ; повторить цикл если они не  равны
     jz ravno
     . . .              ; обработка если числа нет в массиве
     jmp endcmp
ravno:  . . .           ; обработка если число есть в массиве
endcmp: . . .                ; продолжение программы
```

Реализация циклов общего вида

- Используются команды условных переходов.

Пример.

```
While (A!=B) {
   // Тело цикла
}
```

```
BegLoop:    cmp AX, BX
     jz EndLoop
     . . .      ; Тело цикла
     jmp BegLoop
EndLoop:    . . .
```

Реализация «длинных» циклов

- В «длинном цикле» переход выполняется на смещение, превышающее диапазон -128..+127 байт.

Пример.

```
    mov CX, Count
L1:     . . .           ; Тело цикла
    dec CX
    jcxz L2
    jmp L1
L2:     . . .           ; Цикл завершен
```

Самостоятельная работа

Задание 1.

Дано 16-разрядное битовое поле (регистр). Реверсировать порядок битов.

Задание 2.

Дан массив из 10 знаковых чисел (слов). Найти минимальный и максимальный элементы массива.

Команды умножения

mul множитель ; умножение беззнаковых чисел

imul множитель ; умножение знаковых чисел

- длины множимого и множителя должны быть равны:
- в команде указывается только множитель, который может быть или регистром или ячейкой памяти;
- множимое всегда находится в аккумуляторе (AX):
- для записи произведения (результата) используется в 2 раза больше байт, чем у множителя.

Операнд	Действие	Результат	Расширение
байт	AL * операнд	AX	AH
слово	AX * операнд	DX,AX	DX
двойное слово	EAX * операнд	EDX, EAX	EDX

Команды умножения

- После MUL флаги CF и OF равны нулю, если старшая половина произведения равна 0, в противном случае оба флага равны 1.
- После IMUL флаги CF и OF равны нулю, если старшая половина содержит только расширение знака, в противном случае оба флага равны 1.
- Остальные флаги после этих команд неопределенны.

Ограничения.

- В командах нельзя указывать непосредственный операнд – его нужно предварительно загрузить в регистр или ячейку памяти.

```
mov bx, 10
mul bx
```

- Длина операндов при умножении должна быть равной. При умножении операндов разной длины меньший нужно расширить до длины большего. При беззнаковом умножении нулями, при знаковом – командами знакового расширения или movsx.

```
mov AL, bb
cbw             ;
mul ww          ; mul ww
```

Команды умножения. Примеры

Пример 1.

mov AL, 37

mov BL, 5

imul BL

AX будет содержать 00B9h (+185), при этом CF = 0 и OF = 0, т.к. регистр AH содержит все нули.

Пример 2.

mov AL, -37

mov BL, 5

imul BL

После выполнения операции умножения регистр AX содержит 0FF47h (-185). Поскольку в регистре AH содержится расширение знака регистра AL (0FFh), то флаги имеют нулевые значения: CF = 0, OF = 0.

Команды деления

div делитель ; деление беззнаковых чисел

idiv делитель ; деление знаковых чисел

- Делимое всегда находится в аккумуляторе или аккумуляторе с расширением;
- Делитель задается операндом команды, размер которого в 2 раза меньше размера делимого (в байтах);
- Частное от деления помещается в младшую часть делимого;
- Остаток от деления помещается в старшую часть делимого.

Операнд	Делимое	Частное	Остаток	Диапазон частного
1 байт	AX	AL	AH	div – 0..255 idiv – -128..127
2 байта	DX:AX	AX	DX	div – 0..65535 idiv – -32768..32767
4 байта	EDX:EAX	EAX	EDX	

Команды деления

- Состояние флагов после выполнения деления неопределенно.
- При использовании команд div и idiv может возникнуть переполнение, что вызывает прерывание (деление на ноль, слишком большое частное).
- ??Чтобы избежать переполнения, нужно следовать таким правилу: модуль делителя должен быть меньше модуля старшей части делимого. Эту проверку нужно выполнять перед командой деления. ?? – из литературы
- Модуль делимого ***с учетом старшей части*** должен быть больше модуля делителя.

Операция деления	Делимое	Делитель	Частное
Слово на байт	0123h	01h	(1) 23h
Двойное слово на слово	0001 4926h	01h	(1) 4026h

Команды деления. Проверка

?? **Пример**. Деление беззнаковых чисел.

```
dataseg
DIVISOR DB ?
codeseg
cmp AH, DIVISOR
jb overflow
div DIVISOR
overflow:
< обработчик переполнения> ??
```

Для команды idiv необходимо учитывать, что либо делимое, либо делитель может быть отрицательным, а так как сравниваются абсолютные значения, нужно использовать команду neg для временного преобразования отрицательного значения в положительное.

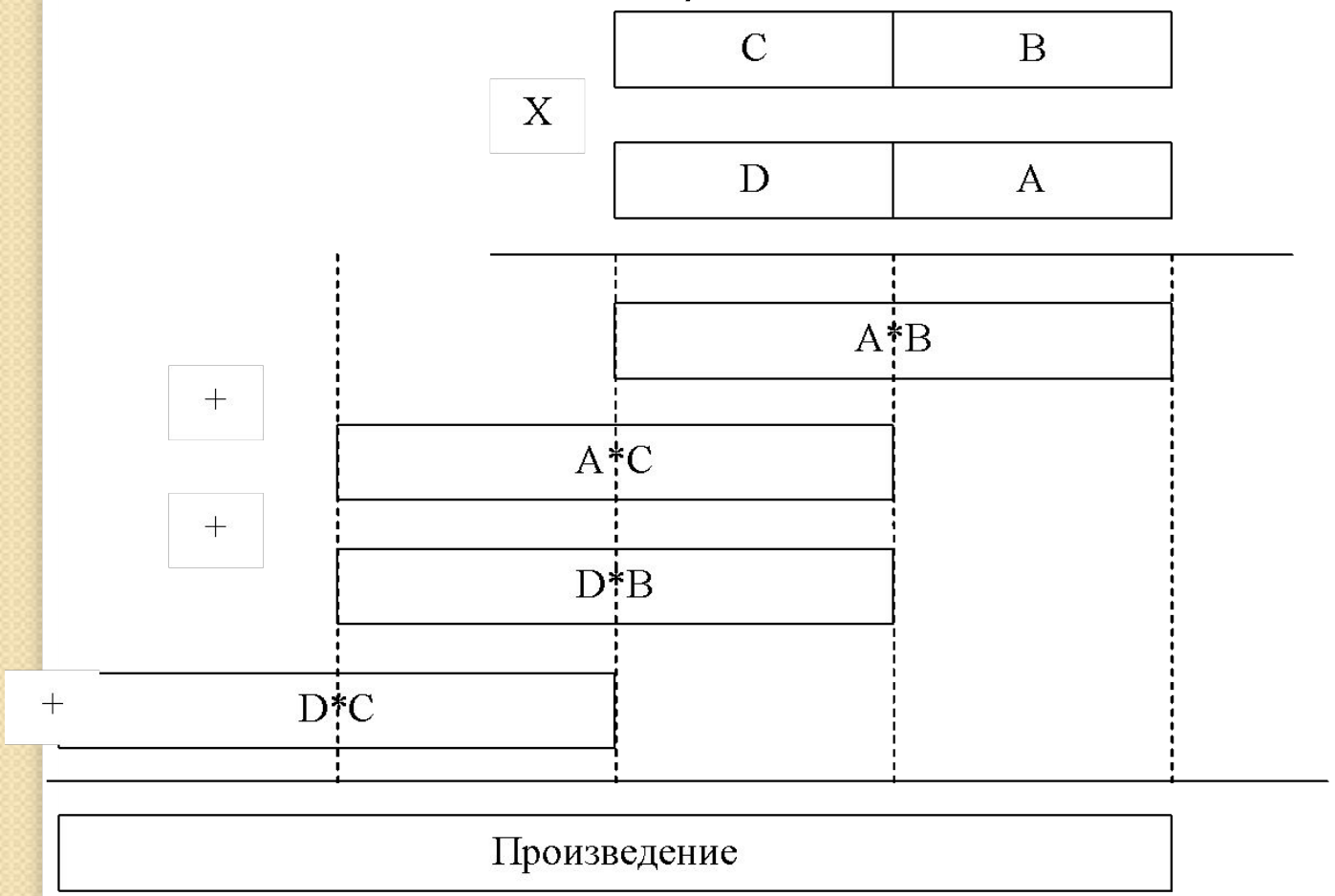
Если при этом отрицательное делимое занимает 2 регистра, преобразование знака нужно выполнять вручную: сначала инвертировать биты, а затем прибавить 1 к полученному числу.

Пример. Преобразование знака делимого в DX:AX.

```
not DX          ; инвертирование битов в DX
not AX          ; инвертирование битов в AX
add AX, l       ; прибавление 1 к AX
adc DX, o       ; прибавление переноса к DX
```

Умножение многоразрядных чисел

- Умножение чисел большой разрядности может привести к появлению результата, разрядность которого не может поместиться в пару регистров EDX:EAX (RDX:RAX).
- В таких случаях умножение может быть реализовано по принципу умножения в столбик по следующей схеме:



Умножение многоразрядных чисел. Пример

Пример. Умножение двух двойных слов с получением 64-разрядного результата с использованием 16-разрядных регистров.

Исходные операнды:

CX, BX – первый множитель,

DX, AX – второй множитель.

Результат записывается в 4 регистра: DX, CX, BX, AX.

```
dataseg
HI_M    dw  ?          ; для сохранения ст. части операнда
LO_M    dw  ?          ; для сохранения мл. части операнда
HI_PP1  dw  ?          ; ст. часть 1 промеж. произведения
LO_PP1  dw  ?          ; мл. часть 1 промеж. произведения
HI_PP2  dw  ?          ; ст. часть 2 промеж. произведения
LO_PP2  dw  ?          ; мл. часть 2 промеж. произведения
HI_PP3  dw  ?          ; ст. часть 3 промеж. произведения
LO_PP3  dw  ?          ; мл. часть 3 промеж. произведения
HI_PP4  dw  ?          ; ст. часть 4 промеж. произведения
LO_PP4  dw  ?          ; мл. часть 4 промеж. произведения
```

Умножение многоразрядных чисел. Пример

```
codeseg
mov LO_M, AX          ; сохраняем мл. часть 2 операнда
mov HI_M, DX          ; сохраняем ст. часть 2 операнда
mul BX              ; DX,AX ← A*B
mov LO_PP1, AX        ; сохраняем произведение
mov HI_PP1, DX        ; --
mov AX, HI_M          ; AX←D
mul BX              ; DX,AX ← D*B
mov LO_PP3, AX        ; сохраняем произведение
mov HI_PP3, DX        ; --
mov AX, LO_M          ; AX←A
mul CX              ; DX,AX ← A*C
mov LO_PP2, AX        ; сохраняем произведение
mov HI_PP2, DX        ; --
mov AX, HI_M          ; AX←D
mul CX              ; DX,AX ← D*C
mov LO_PP4, AX        ; сохраняем произведение
mov HI_PP4, DX        ; --
mov AX, LO_PP1        ; мл. часть результата в AX
mov BX, HI_PP1        ; вычисляем вторую часть результата в BX
add BX, LO_PP2        ; добавляем мл. часть 2 произведения
adc HI_PP2, 0         ; добавляем перенос в ст. часть произведения 2
add BX, LO_PP3        ; добавляем мл. часть 3 произведения
                ; BX готов
```

Умножение многоразрядных чисел. Пример

```
adc HI_PP3, 0        ; добавляем перенос в ст. часть произведения 3
mov CX, HI_PP2           ; вычисляем третью часть произведения в CX
add CX, HI_PP3       ; добавляем ст. часть произв.3
adc HI_PP4, 0;       ; добавляем перенос к ст. части произв.4
add CX, LO_PP4           ; CX готов
mov DX, HI_PP4           ; DX←ст. часть произв.4
adc DX, 0         ; DX готов
```

Двоично-десятичная арифметика

Форматы представления двоично-десятичных чисел:

- Числа в формате ASCII,
- Неупакованные двоично-десятичные числа (BCD-числа),
- Упакованные двоично-десятичные числа.

Формат ASCII:

- ввод чисел с консоли или вывод на какое-либо устройство (дисплей или принтер);
- старший (левый) полубайт каждого байта содержит значение 3h;
- младший (правый) полубайт — значение десятичного разряда.

Пример. 6591 - 36353931h.

Неупакованный двоично-десятичный формат.

- левые полубайты таких чисел установлены в 0;
- операции выполняются медленнее, чем над двоичными числами.
- можно легко организовать обработку чисел большой разрядности.

Пример. 6591 - 06050901h.

Арифметика BCD и ASCII-чисел. Сложение

Сложение одноразрядных ASCII чисел выполняется в 3 этапа:

- Сложение командой add/adc, результат должен быть в AX.
- Коррекция регистра AX командой aaa. Результат – в AX правильное неупакованное двузначное BCD-число.
- Установка значения 3 в старшие полубайты AX.

Пример.

```
dataseg
num1   db 34h
num2   db 38h
codeseg
mov AL, num1
mov BL, num2
add  AL, BL
aaa                ; AX←0102h
or AX, 3030h  ; AX←3132h
```

Арифметика BCD и ASCII-чисел. Сложение

Для реализации сложения многоразрядных ASCII-чисел нужно организовать цикл, складывающий соответствующие разряды от младших к старшим с учетом переноса.

```
dataseg
num1  db '0037'
num2  db '0986'
len     dw 4
sum         db 4 dup (?)
codeseg
    mov CX, len
    clc             ; Очистка флага переноса
    mov BX, CX
begin: dec BX       ; Смещение последней складываемой цифры
    mov AL, byte ptr num1[BX]
    adc AL, byte ptr num2[BX]   ; сложение двух ASCII-цифр
    aaa                         ; коррекция AX
    mov byte ptr sum[BX], AL    ; запись рез-та в соотв. Байт
    loop begin
    or dword ptr sum, 30303030h   ; переход к ASCII-числу в
SUM
```

Арифметика BCD и ASCII-чисел. Вычитание

Вычитание одноразрядных ASCII чисел выполняется в 3 этапа:

- Вычитание командой sub/sbb, результат должен быть в AX.
- Коррекция регистра AX командой aas. Результат – в AX правильное неупакованное двузначное BCD-число.
- Установка значения 3 в старшие полубайты AX.

Пример.

```
dataseg
num1   db '4'
num2   db '8'
codeseg
mov AL, num1
mov BL, num2
sub  AL, BL
aas          ; AX←FF04h – отрицательный результат
or AX, 3030h  ; AX←3F34h
```

Для реализации вычитания многоразрядных ASCII-чисел нужно организовать цикл, вычитающий соответствующие разряды от младших к старшим с учетом флага переноса (заема).

Арифметика BCD и ASCII-чисел. Умножение

Умножение одноразрядных ASCII чисел выполняется в 4 этапа:

- Преобразование ASCII-чисел в BCD-числа.
- Умножение командой mul, результат должен быть в AX.
- Коррекция регистра AX командой aam. Результат – в AX правильное неупакованное двузначное BCD-число.
- Установка значения 3 в старшие полубайты AX.

Пример.

```
dataseg
num1    db '4'
num2    db '8'
codeseg
mov AL, num1
mov BL, num2
and AL, 0Fh
and BL, 0Fh
mul  BL
aam              ; AX←0302h –результат
or AX, 3030h     ; AX←3332h
```

Для реализации умножения многоразрядных ASCII-чисел нужно организовать цикл умножения «в столбик» с получением промежуточных произведений и их последующим сложением.

Арифметика BCD и ASCII-чисел. Деление

Деление одноразрядных ASCII чисел выполняется в 4 этапа:

- Преобразование ASCII-чисел в BCD-числа.
- Коррекция двухбайтового делимого в регистре AX командой aad.
- Деление командой div. Результат – в AL неупакованное двузначное BCD-число – частное, в AH неупакованное BCD-число – остаток.
- Установка значения 3 в старшие полубайты AX.

Пример.

```
dataseg
numl    db '34'
num2    db '8'
codeseg
mov AX, num1
mov BL, num2
and AX, 0F0Fh
and BL, 0Fh
aad
div  BL        ; AL←04h, AH←02h
or AX, 3030h    ; AX←3234h
```

Деление многоразрядных чисел выполняется методом «в столбик» или последовательностью вычитаний.

Арифметика упакованных чисел. Сложение

С упакованными двоично-десятичными числами можно выполнять только операции сложения и вычитания, после которых необходимо выполнить коррекцию.

daa – десятичная коррекция для сложения. Преобразует двоичный результат выполнения команд add и adc в регистре AL в упакованное десятичное число.

Пример.

```
    dataseg
op1    db  32h
op2    db  59h
    codeseg
    mov AL, op1
    add    AL, op2    ; AL←8Bh
    daa               ; AL←91h
```

Если после выполнения команды daa флаг CF=1, произошел перенос единицы из старшего разряда.

Арифметика упакованных чисел. Вычитание

das – десятичная коррекция для вычитания. Преобразует двоичный результат выполнения команд sub и sbb в регистре AL в упакованное десятичное число.

Пример.

```
op1    db  32h
op2    db  59h
    codeseg
    mov AL, op2
    subAL, op1     ; AL←27h
    das            ; AL←27h
```

Если после выполнения команды das флаг CF=1, произошел заем единицы в старший разряд.

Команды модификации флагов

Изменение флага CF

CLC – обнулить флаг CF: CF←0.
STC – установить флаг CF: CF←1.
CMC – инвертировать флаг CF: CF←CF xor 1.

Флаг направления DF

Используется при обработке строк и определяет направление обработки (0– обработка от меньших адресов к большим, 1 – наоборот).
CLD – обнулить флаг DF: DF←0.
STD – установить флаг DF: DF←1.

Флаг прерывания IF

Определяет реакцию системы на прерывания от внешних устройств (0– прерывания игнорируются, 1– процессор реагирует на прерывания).
CLI – обнулить флаг IF: IF←0.
STI – установить флаг IF: IF←1.

СИМВОЛЬНАЯ ОБРАБОТКА

1. Преобразование двоичных чисел при вводе
2. Преобразование двоичных чисел при выводе

3. Преобразование десятичных чисел при вводе
4. Преобразование десятичных чисел при выводе

Преобразование двоичных чисел при вводе

- Для переменной размером 1 байт:

```
for i:=1 to 8
        if  [вх_буфер] = ‘0’ then CF=0;
        else CF = 1;
        rcl результат, 1;
        вх_буфер++;
end for
```

AL – регистр для ввода;
DI – адрес начала вх_буфера;
CX – счетчик;

```
        mov CX, 8
st_loop:  cmp byte ptr [DI], ‘0’
        je C1
        stc
        jmp e_loop
C1:  clc
e_loop:   rcl AL, 1
        inc DI
        loop st_loop
```

Преобразование двоичных чисел при выводе

- Для переменной размером 1 байт:

```
for i:=1 to 8
      сдвиг_влево на 1;
      if CF=1 then [вых_буфер] = ‘1’;
      else [вых_буфер] = ‘0’;
      вых_буфер++;
end for
```

AL – число для вывода;
DI – адрес начала вых_буфера;
CX – счетчик;

```
      mov CX, 8
st_loop:  rol AL, 1
      jc C1
      mov byte ptr [DI], ‘0’
      jmp e_loop
C1:  mov byte ptr [DI], ‘1’
e_loop:   inc DI
      loop st_loop
```

Преобразование десятичных чисел при вводе

- Знаковое число от -32768 до +32767 (16 битов). Число символов = длине буфера ввода. Проверка на ошибку не выполняется

```
признак = 0;
if [вх_буфер]=='-' then
        признак = 1;
        вх_буфер++;   длина_строки--;
else   if [вх_буфер]=='+' then
                признак = 0;
                вх_буфер++;   длина_строки--;
        end if
end if
```

```
результат = 0;
for i=1 to длина_строки
        результат  = результат * 10;
        результат = результат +
                + ASCII_2_BIN([вх_буфер]);
        вх_буфер++;
end for
```

```
if признак=1 then
        результат = - результат;
end if
```

AX – регистр для ввода;
SI – адрес начала вх_буфера;
CX – число символов во вх_буфере;
BX= 10 – константа для умножения;
DI – признак;

```
        mov DI, 0
        cmp byte ptr [SI], '-'
        je neg_v
        cmp byte ptr [SI], '+'
        je pos_v
        jmp st_digit
neg_v:  mov DI, 1
        dec CX
        inc SI
        jmp st_digit
pos_v:  mov DI, 0
        dec CX
        inc SI
```

```
st_digit:   mov AX, 0
st_loop:    mul BX;   рез=DX:AX
        mov DH, 0
        mov DL, [SI]
        and DL, 0Fh
        add  AX, DX
        inc SI
        loop st_loop
```

```
        cmp DI, 1
        jne end_w
        neg AX
end_w:  ...
```

Преобразование десятичных чисел при выводе

- Знаковое число от -32768 до +32767 (16 битов). Число знакомест = 6 (с учетом возможного знака). Ведущие нули не выводятся

```
for i=1 to 6
        [вых_буфер] = ‘ ‘;
        вых_буфер++;
end for
вых_буфер--;
```

```
признак = 0;
if  число<0 then
        признак = 1;
        число = -число;
end if
```

```
do
        (число, остаток) = число /10;
        [вых_буфер] = BIN_2_ASCII(остаток);
        вых_буфер--;
while (число<>0);
```

```
if признак=1 then
        [вых_буфер] = ‘-’;
end if
```

AX – число для вывода;
SI – адрес начала вых_буфера;
BX= 10 – константа для деления;
DI – признак;

```
mov CX, 6
cl_field:       mov byte ptr [SI], ‘ ‘
        inc SI
        loop cl_field
        dec SI
```

```
        mov DI, 0
        cmp AX, 0
        jg st_loop
        mov DI, 1
        neg AX
```

```
st_loop:        xor DX, DX
        div BX;   рез=DX,AX
        add DL, ‘0’
        mov [SI], DL
        dec SI
        cmp AX, 0
        jz end_w
        jmp st_loop
```

```
end_w:          cmp DI, 1
        jne end_w1
        mov byte ptr [SI], ‘-’
end_w1:         ...
```

Обработка строк

Понятие строки

Строка – непрерывная область памяти, длиной:

- В реальном режиме – до 64K:
- В защищенном режиме – неограниченной длины.

Строки в языках программирования высокого уровня:

- **Short string** (короткая строка) – используется в языке Pascal и системе Delphi. Первый байт строки содержит количество символов строки, а последующие – сами символы. Длина такой строки – от 0 до 255 символов.
- **Null-terminated string** – строка, конец которой обозначается символом с кодом 0. Такие строки наиболее распространены (Си и др.).

Для ассемблера:

- Строка – это последовательность байт, начинающихся с заданного адреса.
- Элемент строки может быть размером 1, 2 или 4 байта.
- Конкретный вид строки и порядок обработки задает сам программист.

Пример.

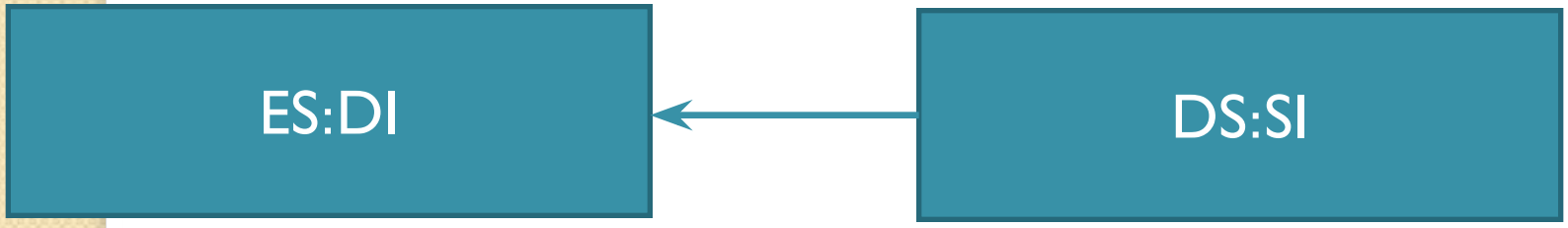
```
        dataseg
ShortString db 6,'Строка'    ; Короткая строка
NullTermStr     db 'Строка', 0   ; Null-terminated string
```

Цепочечные примитивы

- Цепочечный примитив – это команда, предназначенная для обработки одного элемента строки (массива).
- Отдельный примитив обрабатывает один элемент строки.

В общем случае, примитивы работают с 2 областями памяти.

- Область, из которой данные поступают на обработку, является источником.
- Источник всегда адресуется парой регистров DS:SI.
- Область, в которую данные помещаются после обработки, является приемником.
- Приемник всегда адресуется парой регистров ES:DI.
- В некоторых примитивах используется только источник или только приемник.



Инкремент или декремент

- После выполнения любого из примитивов содержимое индексных регистров DI и SI автоматически увеличивается или уменьшается на одну и ту же величину – величину длины обрабатываемого элемента строки (1, 2 или 4).
- Направление изменения (увеличение или уменьшение) зависит от значения флага DF

Элемент строки	Флаг DF	
	0	1
Байт	+1	-1
Слово	+2	-2
Двойное слово	+4	-4

Примитивы 1

Копирование строк

- movsb – копирование байта
- movsw – копирование слова
- movsd – копирование двойного слова
- mem[ES:DI] ← mem[DS:SI]
- Флаги не модифицируются.

Сравнение строк

- cmpsb – сравнение байт
- cmpsw – сравнение слов
- cmpsd – сравнение двойных слов
- mem[ES:DI] - mem[DS:SI]
- Флаги модифицируются аналогично команде cmp

Примитивы 2

Сканирование строк

- scasb – сканирование байт
- scasw – сканирование слов
- scasd – сканирование двойных слов
- сравнение аккумулятора (AL, AX, EAX) с элементом строки по адресу ES:DI
- аккумулятор - mem[ES:DI]
- Флаги модифицируются аналогично команде cmp

Загрузка строк

- lodsb – загрузка байта
- lodsw – загрузка слова
- lodsd – загрузка двойного слова
- Копирование элемента строки из памяти в аккумулятор (AL, AX, EAX)
- аккумулятор ← mem[DS:SI]
- Флаги не модифицируются.

Примитивы 3

Выгрузка строк

- stosb – выгрузка байта
- stosw – выгрузка слова
- stosd – выгрузка двойного слова
- копирование аккумулятора (AL, AX, EAX) в элемент строки
- mem[ES:DI] ← аккумулятор
- Флаги не модифицируются.

Пример. Заполнение области памяти определенным символом.

```
        dataseg
s1      db  dup 20 (?)
len     dw 20
        codeseg
        mov AX, @DATA
        mov ES, AX
        cld             ; обнулить DF для инкремента адреса
        mov AL, 'X'  ; заполнитель
        lea  DI, s1     ; адрес строки приемника
        mov CX, len; количество символов
l:      stosb                   ; заполнить строку заполнителем
        loop l
```

ПРЕФИКСЫ ПОВТОРЕНИЯ

- Префикс повторения обеспечивает выполнение одного цепочечного примитива несколько раз.
- Количество повторений определяется содержимым счетчика CX.
- Задает цикл из одной команды – цепочечного примитива

Префикс	Действие	Цеп. Примитив
rep	выполнять пока CX<>0	movs, lods, stos
repe, repz	выполнять пока CX<>0 и ZF=1	cmps, scas
repne, repnz	выполнять пока CX<>0 и ZF=0	cmps, scas

Пример. В предыдущем примере цикл loop можно заменить командой:

```
rep stosb
```

ОТЛИЧИЕ REP ОТ LOOP

- Префикс повторения используется только с цепочечным примитивом;
- CX проверяется до выполнения примитива, т.е. реализуется цикл с предусловием.
- Эквивалентная rep movsb запись:

```
l1:jcxz   l2
   mov AL, [SI]
   mov [DI], AL
   inc/dec DI
   inc/dec SI
   dec CX
   jmp l1
l2:
```

ПРИМЕР 1

Подсчет количества слов во фрагменте текста.

```
        dataseg
s1      db    ‘  text   string  for  example  $‘
len     dw    32
        codeseg
        mov AX, @DATA
        mov DS, AX
        mov ES, AX
        mov CX, len         ; размер строки
        lea DI, s1     ; адрес первого символа строки
        mov AL, ‘ ‘  ; разделитель слов
        xor BX, BX; счетчик слов
        cld
next:
        repe scasb          ; пропускаем пробелы
        je exit             ; кроме пробелов ничего нет – закончить
        inc BX              ; нарастить счетчик
        repne scasb; ищем конец слова
        jne exit            ; строка закончилась – закончить
        jmp next
exit:                       ; BX – счетчик слов
```

ПРИМЕР 2

Сравнение двух строк.

```
        dataseg
s1      db    ‘  text   string  for  example‘,0
S2      db    ‘  text   string  for’,0
len     dw    31
        codeseg
        mov AX, @DATA
        mov DS, AX
        mov ES, AX
        mov CX, len        ; размер строки
        lea SI, s1          ; адрес первой строки
        lea DI, s2    ; адрес второй строки
        cld           ; прямое направление обработки строк
        repe cmpsb; сравниваем строки
        jne mithmatch     ; строки не равны
match       . . .         ; строки равны - обработка
        jmp exit
Mithmatch . . .           ; строки не равны - обработка
        ; jb less1        ; s1 меньше
        ; ja great1    ; s1 больше
        . . .
exit:
```

Подпрограммы

Понятие подпрограммы

Вызов ПП заключается в передаче управления в новую точку сегмента кода и запоминании адреса точки вызова, для того чтобы после окончания работы ПП можно было бы вернуться к следующей за ней команде.

Преимущества использования подпрограмм:

- Позволяют разбить программу на логически законченные части;
- Позволяют сократить длину программы при многократном использовании повторяющейся последовательности действий;
- Упрощают внесение изменений в программный код.
- Позволяют создавать библиотеки подпрограмм и использовать их при разработки различных программ.

СХЕМА ПРОГРАММЫ С ПОДПРОГРАММАМИ

```
        mov AX, 0       ; Начальные значения регистров
        mov     BX, 0
        . . .
        jmp  Start      ; Переход к выполнению основной программы
add1    proc    ; Точка входа в процедуру add1
        inc AX          ; Команды процедуры
        ret     ; Возврат в вызывающую программу
add1    endp            ; Конец описания процедуры
sub1    proc    ; Описание процедуры sub1
        dec BX
        ret
sub1    endp
start:
        call  add1              ; вызов процедуры add1
        call  sub1      ; вызов процедуры sub1
        . . .
```

Директивы описания подпрограммы

Описание начала процедуры:

ИмяТочкиВхода proc [near/ far]

- near указывает на то, что процедура является ближней. К ближней процедуре можно обращаться только из того сегмента команд, где она объявлена;
- far указывает на то, что процедура дальняя. К дальней процедуре можно обращаться из любых сегментов команд, включая тот, где она объявлена;
- если тип процедуры отсутствует, считается, что она имеет тип near;
- для 32-разрядных приложений с моделью flat все вызовы процедур считаются ближними.

Описание конца процедуры:

ИмяТочкиВхода endp

Вызов процедуры

call ИмяПроцедуры

В момент вызова процедуры команда call помещает в стек адрес команды, следующей непосредственно за call, уменьшая значение указателя стека SP (ESP).

Команда call может иметь один из следующих форматов вызова:

- прямой ближний (в пределах текущего программного сегмента);
- прямой дальний (вызов процедуры, расположенной в другом программном сегменте);
- косвенный ближний (в пределах текущего программного сегмента с использованием переменной, содержащей адрес перехода);
- косвенный дальний (вызов процедуры, расположенной в другом программном сегменте, с использованием переменной, содержащей адрес перехода).

Тип команды call, используемый по умолчанию, зависит от применяемой модели памяти:

- tiny, small, compact, flat – near,
- medium, large и huge – far.

Способы вызова CALL

Прямой вызов – вызов по метке точки входа:

- call ИмяТочкиВхода ; Тип вызова определяется по умолчанию
- call near ptr ИмяТочкиВхода ; Ближний вызов
- call far ptr ИмяТочкиВхода ; Дальний вызов

Косвенный ближний вызов – вызов по значению адреса в памяти длиной в слово:

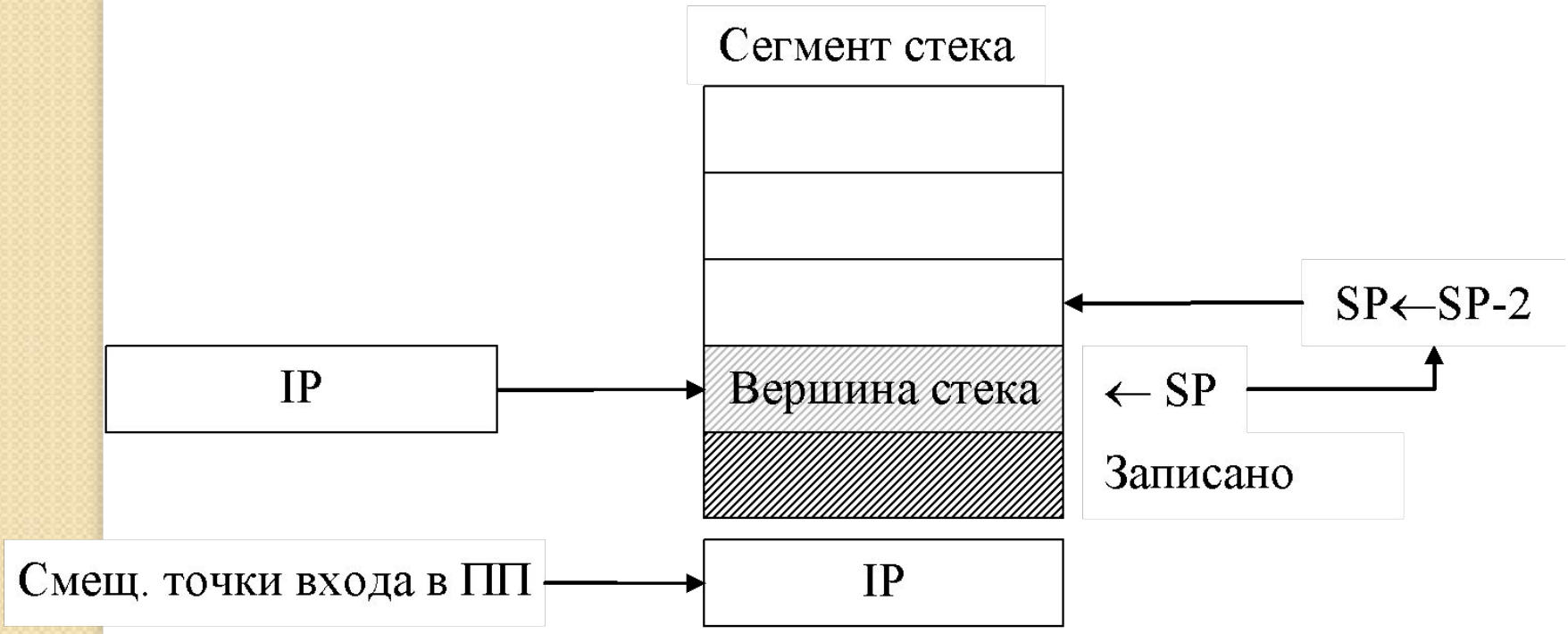
- call word ptr BX ; адрес подпрограммы находится в регистре BX
- call word ptr [BX] ; адрес подпрограммы находится в ячейке памяти,
  ; адрес которой помещается в регистр BX
- call word ptr [BX][SI] ; в BX – адрес таблицы адресов подпрограмм.
  ; в SI индекс в этой таблице
- call word ptr tbl[SI] ; переменная tbl содержит адрес таблицы адресов
  ; подпрограмм, в SI – индекс в этой таблице

Косвенный дальний вызов – вызов по значению адреса в памяти длиной в двойное слово:

- call dword ptr [BX]
- call dword ptr [BX][SI]
- call dword ptr tbl[SI]

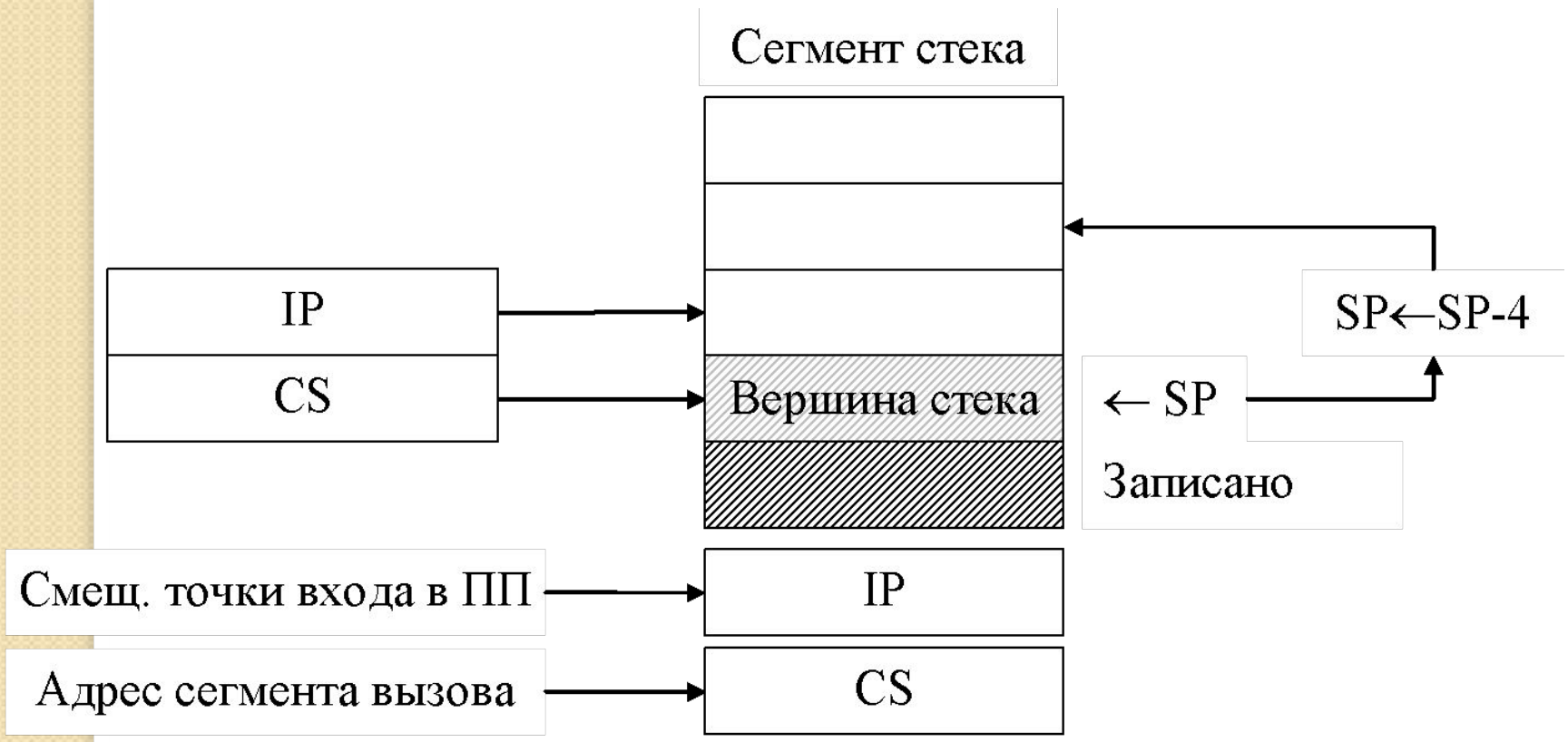
Действие CALL. Ближний вызов

- Помещает в стек относительный адрес точки возврата в текущем программном сегменте (2 байта);
- Модифицирует указатель адресов команд IP так, чтобы в нем содержался относительный адрес точки перехода в том же программном сегменте (адрес первой команды подпрограммы).



Действие CALL. Дальний вызов

- Помещает в стек два слова: вначале сегментный адрес текущего программного сегмента (CS), затем относительный адрес точки возврата в этом программном сегменте (IP).
- Выполняется модификация регистров IP и CS: в IP помещается относительный адрес точки перехода в том сегменте, куда осуществляется переход, а в CS —адрес этого сегмента



Косвенный ближний вызов. Пример

```
                .model small; Вызов процедур из таблицы адресов процедур.
data            segment
tbl             DW subr1            ; смещение процедуры subr1
                DW subr2            ; смещение процедуры subr2
data            ends
code            segment
                assume CS:code, DS:data
main            proc
                mov   AX, data
                mov   DS, AX        ; адрес сегмента данных
                lea   SI, tbl ; адрес первой процедуры
                xor   BX, BX        ; начальное смещение
                mov CX, 2           ; кол-во процедур
next:
                call word ptr [BX][SI]      ; вызов очередной процедуры
                add BX, 2           ; адрес следующей процедуры
                loop next
main            endp                ; конец осн. Процедуры
subr1 proc                  ; описание процедуры subr1
                . . .
subr1 endp
subr2 proc                  ; описание процедуры subr2
                . . .
subr2 endp
```

Косвенный дальний вызов. Пример

```
.model large
data    segment
tbl     DD ?            ; дальний адрес процедуры subr1
        DD ?            ; дальний адрес процедуры subr2
data    ends
code0 segment
        assume CS:code0, DS:data
main    proc
        mov   AX, data
        mov   DS, AX            ; адрес сегмента данных
        lea   SI, tbl           ; адрес первой процедуры
        push SI                 ; сохраняем для дальн. использ.
        mov AX, code1           ; адрес сегмента с процедурами
        mov word ptr [SI], offset subr1  ; смещение процедуры1– первый байт
        mov word ptr [SI+2], AX  ; второе слово – адрес сегмента
        add SI, 4               ; второй элемент таблицы
        mov word ptr [SI], offset subr2  ; смещение процедуры2– первый байт
        mov word ptr [SI+2], AX  ; второе слово – адрес сегмента
        pop SI
```

Косвенный дальний вызов. Пример

```
                xor   BX, BX            ; начальное смещение
                mov CX, 2         ; кол-во процедур
next:
                call dword ptr [BX][SI]       ; дальний вызов очередной процедуры
                add BX, 4               ; адрес следующей процедуры
                loop next
main            endp                    ; конец осн. Процедуры
                . . .
code0                ends
code1           segment
                assume CS:code1
subr1           proc  far               ; описание процедуры subr1
                . . .
subr1           endp
subr2           proc  far               ; описание процедуры subr2
                . . .
subr2           endp
                . . .
code1           ends
```

Возврат из процедуры

RET	**N**	Возврат из процедуры
RETN	**N**	Возврат из ближней процедуры
RETF	**N**	Возврат из дальней процедуры

- Команда ret извлекает из стека адрес возврата и передает управление назад в программу, первоначально вызвавшую процедуру.
- Если командой ret завершается ближняя процедура, объявленная с атрибутом near, или используется модификация команды retn, со стека снимается одно слово- относительный адрес точки возврата
- Если командой ret завершается дальняя процедура, объявленная с атрибутом far, или используется модификация команды retf, со стека снимаются два слова: смещение и сегментный адрес точки возврата.
- Необязательный операнд (кратный 2 указывает, на сколько байтов дополнительно смещается указатель стека после возврата в вызывающую программу. Прибавляя эту константу к новому значению SP, команда ret освобождает из стека параметры, помещенные в него вызывающей программой перед вызовом call.
- Команда не воздействуют на флаги процессора.

Передача параметров и возврат результата

Способы передачи параметров в процедуру:

- через регистры,
- через стек,
- с использованием глобальных переменных,
- с использованием таблицы параметров.

Конкретный способ передачи выбирает программист.

Важно чтобы обработка параметров в ПП была согласована со способом задания параметров в вызывающей программе.

Передача параметров через регистры

- Преимущество – высокая эффективность.
- Недостаток – ограниченное количество регистров процессора.
- Вызывающей программе и процедуре могут одновременно потребуются для работы одни и те же регистры. Нужно сохранить регистры в стеке при входе в подпрограмму и восстановить их при выходе из подпрограммы.
- Можно сохранять и восстанавливать отдельные регистры командами PUSH и POP. Но рекомендуется сохранять и восстанавливать все регистры процессора: PUSHA, POPA.

Пример.

```
    dataseg
X   dw ?
Y   dw ?
Z   dw ?
    codeseg
    pusha               ; сохранить регистры
    mov   AX, X             ; загрузить в них параметры
    mov   BX,Y
    call  near ptr sum
    mov   z, AX         ; получить возвращаемые значения
    popa                ; восстановить регистры
. . .
sum proc  near
    add AX, BX
    ret
sum endp
```

Передача параметров через стек

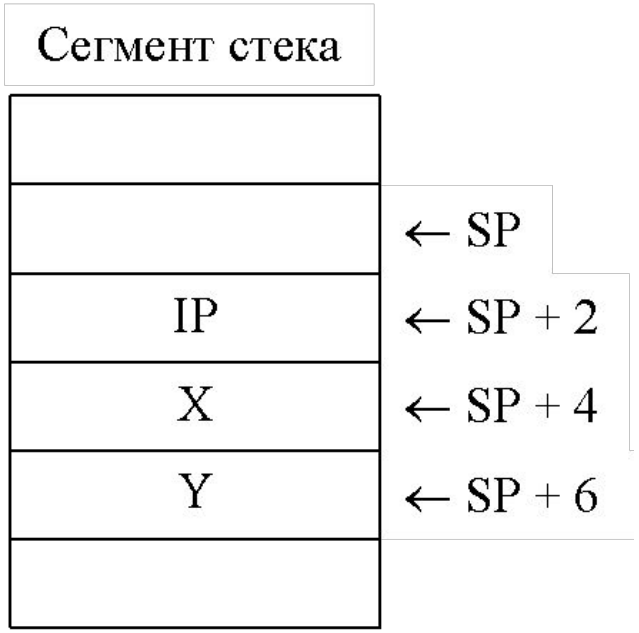
- Основная программа записывает фактические параметры (их значения или адреса) в стек, а процедура затем их оттуда извлекает.
- Наиболее универсальный способ, который используется в большинстве случаев, в том числе и в языках высокого уровня.
- Позволяет передавать неограниченное количество параметров.
- Для доступа к параметрам в стеке используется регистр BP. В него необходимо поместить адрес вершины стека (на него указывает регистр SP), а затем использовать выражения вида [BP+n] для доступа к параметрам процедуры.
- При ближнем вызове адрес первого параметра – [BP+4], т.к. за вершиной стека сохранен IP длиной 2 байта.
- При дальнем вызове адрес первого параметра – [BP+6], т.к. за вершиной стека сохранены IP и CS длиной 4 байта. При этом следует сохранить регистр BP, поскольку он может потребоваться в основной программе.

Передача параметров через стек. Пример 1

Пример 1. С сохранением регистров

```
        dataseg
X       dw ?
Y       dw ?
Z       dw ?
        codeseg
        pusha
        push y          ; параметры в стек
        push x
        call near ptr sum
        mov z, AX       ; результат

        add sp, 4       ;  убрать  параметры  из стека
        popa
        . . .
sum     proc  near
        mov BP, SP      ; BP – вершина стека
        mov AX, [BP+4]      ; извлечение параметров из стека
        add AX, [BP+6]
        ret
sum     endp
```



Передача параметров через стек. Пример 2

Пример 2. Без сохранения регистров

```
dataseg
X    dw ?
Y    dw ?
Z    dw ?
     codeseg
     push    y          ; параметры в стек
     push x
     call near ptr sum
     mov z, AX          ; результат
     . . .
sum     proc     near
     push BP              ; сохранить BP
     mov BP, SP      ; BP – вершина стека
     mov AX, [BP+4]          ; извлечение параметров из стека
     add AX, [BP+6]
     pop BP             ; восстановить BP
     ret 4              ; возврат с удалением 4 байт из стека
sum     endp
```

Передача параметров через глобальные переменные

- Передача значений заключается в обращении к имени глобальной переменной непосредственно из процедуры.
- Глобальные, переменные позволяют работать с данными при минимальном использовании стека, что экономит процессорное время.
- Недостаток – подпрограмма «привязывается» к конкретному набору глобальных переменных.

Пример.

```
dataseg
X   dw ?
Y   dw ?
Z   dw ?
codeseg
    call near ptr sum
     . . .
sum proc near
    push AX
    mov AX, X
    add AX,Y
    mov Z,AX
    pop AX
    ret
sum endp
```

Передача параметров через таблицу параметров

В памяти заводится таблица, и вызывающая программа передает в ПП адрес этой таблицы.

Пример.

```
dataseg
X       dw ?
Y       dw ?
Z       dw ?
T_ARG     dw ?, ?, ?
codeseg
        mov T_ARG, X
        mov T_ARG+2, Y
        mov BX, offset T_ARG
        call near ptr sum
                ...
sum   proc near
        push AX,
        mov AX, [BX]
        add AX, [BX+2]
        mov [BX+4], AX
        pop AX
        ret
sum   endp
```

Структура для доступа к параметрам

ИмяСтруктуры STRUC

директивы db, dw, dd

ИмяСтруктуры ENDS

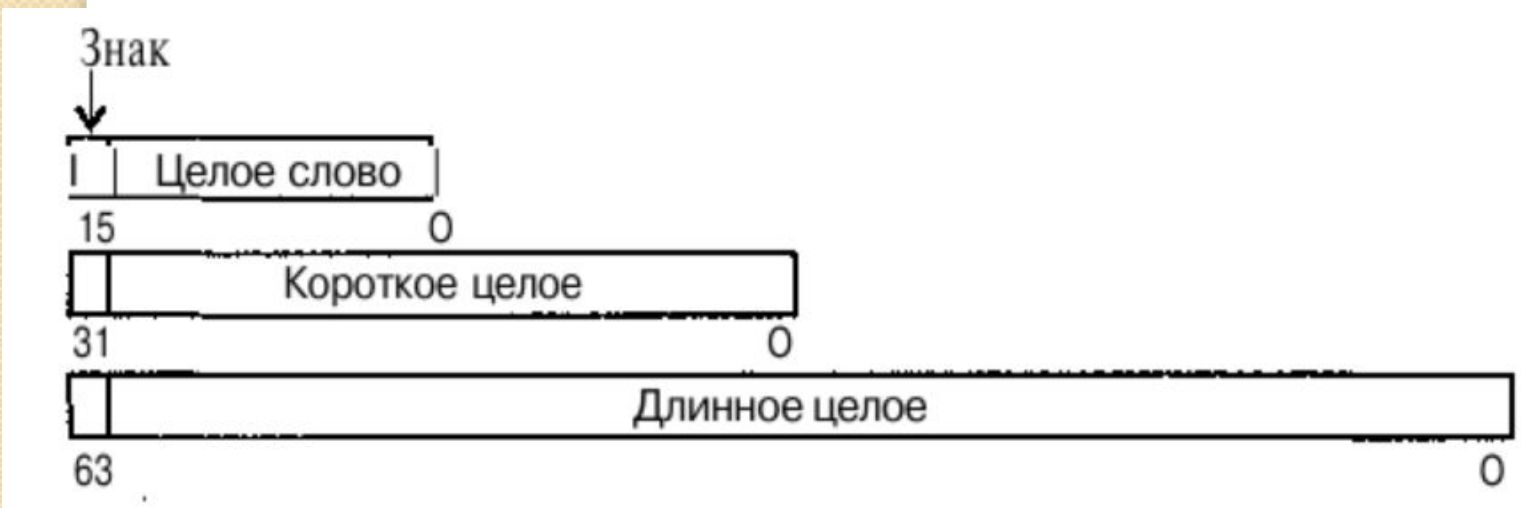
- Описывает структуру параметров процедуры, занесенных в стек при ее вызове.
- Поля структуры могут быть использованы в качестве смещений относительно вершины стека при обращении к параметрам.

Пример.

```
sum  proc near
arg  struc
         sav_bp    dw ?
         sav_ip    dw ?
         slag1  dw ?
         slag2 dw ?
arg  ends
     push BP
     mov BP, SP
     mov AX, [BP]slag1
     add AX, [BP]slag2
     pop BP
     ret 4
sum  endp
```

ДВОИЧНЫЕ ЦЕЛЫЕ ЧИСЛА

Формат	Размер (байт)	Диапазон
Целое слово	2	-32768..32767
Короткое целое	4	$-2 \cdot 10^9 .. 2 \cdot 10^9$
Длинное целое	8	$-9 \cdot 10^{18} .. 9 \cdot 10^{18}$



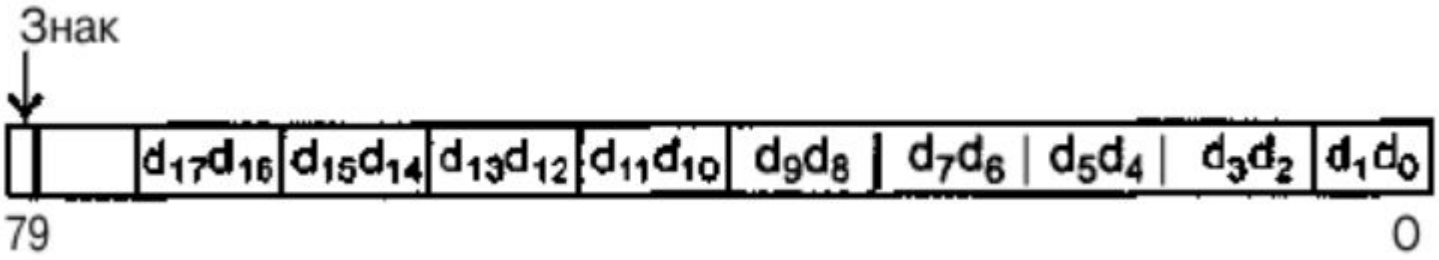
- Сопроцессор переводит целые числа в вещественный формат и обрабатывает в вещественном формате.
- Целые числа могут обрабатывать только команды, второй буквой которых является I.

УПАКОВАННЫЕ ДВОИЧНО-ДЕСЯТИЧНЫЕ ЧИСЛА

- Содержат не более 18 цифр.
- Задаются директивой описания данных DT (10 байт).
- Старший байт этого поля игнорируется.
- Старший бит этого байта хранит знак числа.
- Сопроцессор может только загрузить и выгрузить числа в упакованном формате, вся обработка ведется в вещественном формате.

Пример.

DT $d_{17}d_{16}d_{15}d_{14}d_{13}d_{12}d_{11}d_{10}d_9d_8d_7d_6d_5d_4d_3d_2d_1d_0$



ВЕЩЕСТВЕННЫЕ ЧИСЛА

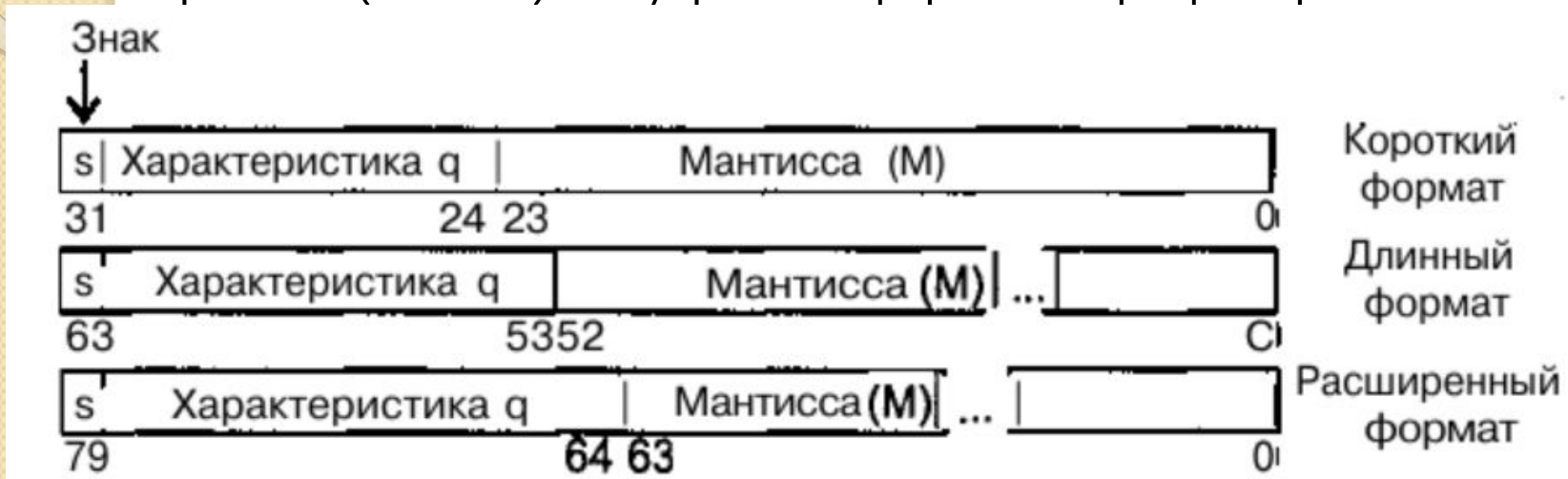
- Основной формат сопроцессора.
- Представляются в виде мантиссы (M) и порядка (p):

$$A = (\pm M) \cdot 2^{\pm(p)}.$$

- Мантисса должна быть нормализована, т.е. удовлетворять соотношению $1 \le M < 2$.
  - Следствие нормализованности - в мантиссе всегда есть единичная целая часть.
  - Следствие нормализованности – для любого вещественного числа существует только одно число с нормализованной мантиссой.
- Порядок p представляется неотрицательной характеристикой q:
  - p = q + фиксированное смещение.
- Для каждого вещественного формата установлено свое значение фиксированного смещения.

Форматы вещественных чисел

- Короткий (4 байта).
- Длинный (8 байт).
- Расширенный (10 байт) – внутренний формат сопроцессора..

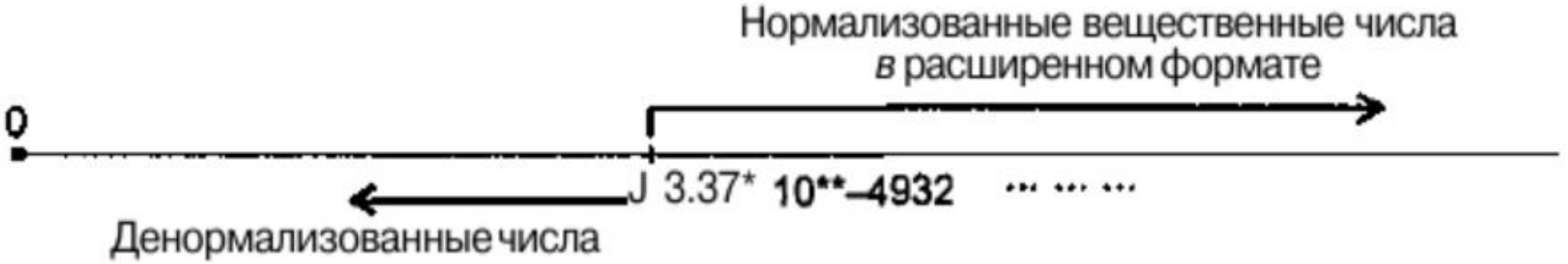


Формат числа	Короткий	Длинный	Расширенный
Диапазон значений	$10^{-38}..10^{38}$	$10^{-308}..10^{308}$	$10^{-4392}..10^{4392}$
Диапазон характеристик	0..255	0..2047	0..32767
Значение фиксированного смещения	-128	-1022	-16382
Диапазон порядков	-128..+127	-1024. . +1023	-16384 .. +16383
Директива описания данных в программе	DD	DQ	DT

СПЕЦИАЛЬНЫЕ ЗНАЧЕНИЯ

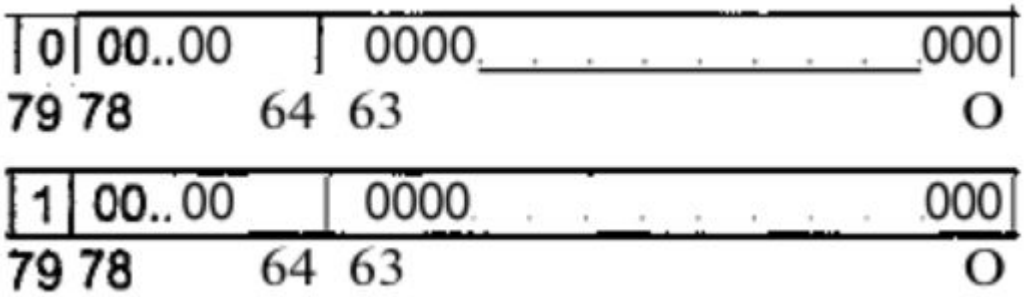
Денормализованные вещественные числа.

- Это числа по модулю меньше минимального нормализованного числа.
- Очень маленькие числа, расположенные между нулем и нормализованными числами.



Нуль.

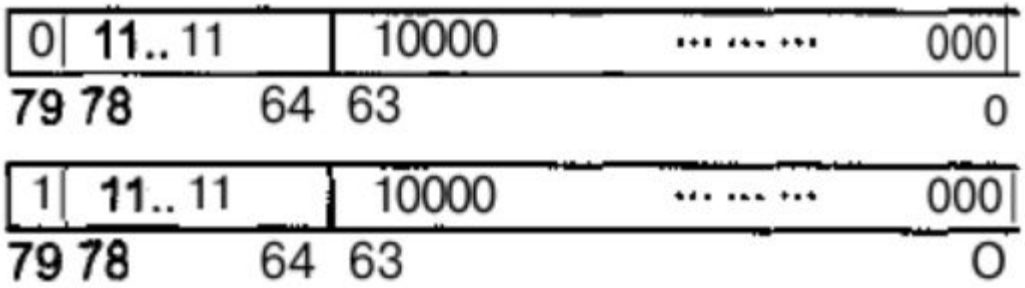
- Может иметь знак – положительный или отрицательный.



СПЕЦИАЛЬНЫЕ ЗНАЧЕНИЯ

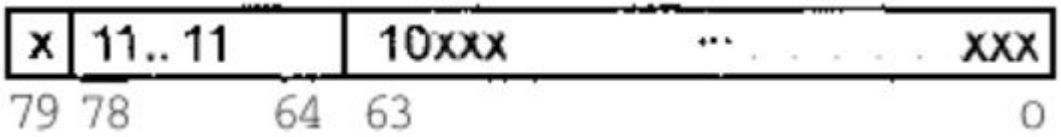
Бесконечность.

- Может иметь знак – положительная или отрицательная.

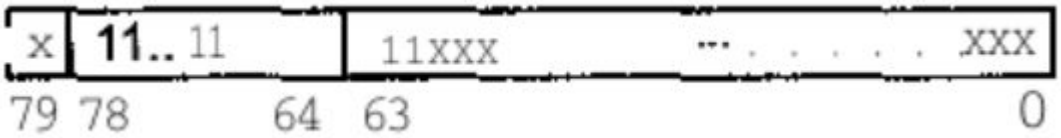


Нечисла.

- Сигнальные нечисла. Сопроцессор не формирует их. Они загружаются программистом для вызова исключительной ситуации.



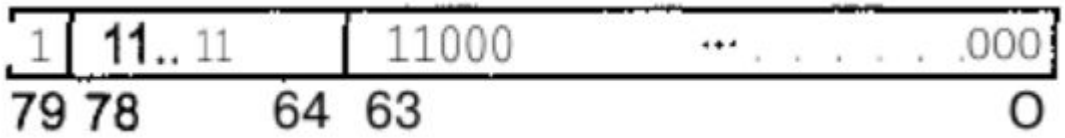
- Тихие (спокойные) нечисла. Формируются сопроцессором при выполнении операции, где один из операндов - тихое нечисло.



СПЕЦИАЛЬНЫЕ ЗНАЧЕНИЯ

Неопределенность.

- Является частным случаем тихого нечисла
- Формируется как маскированная реакция сопроцессора на исключение недействительной операции.



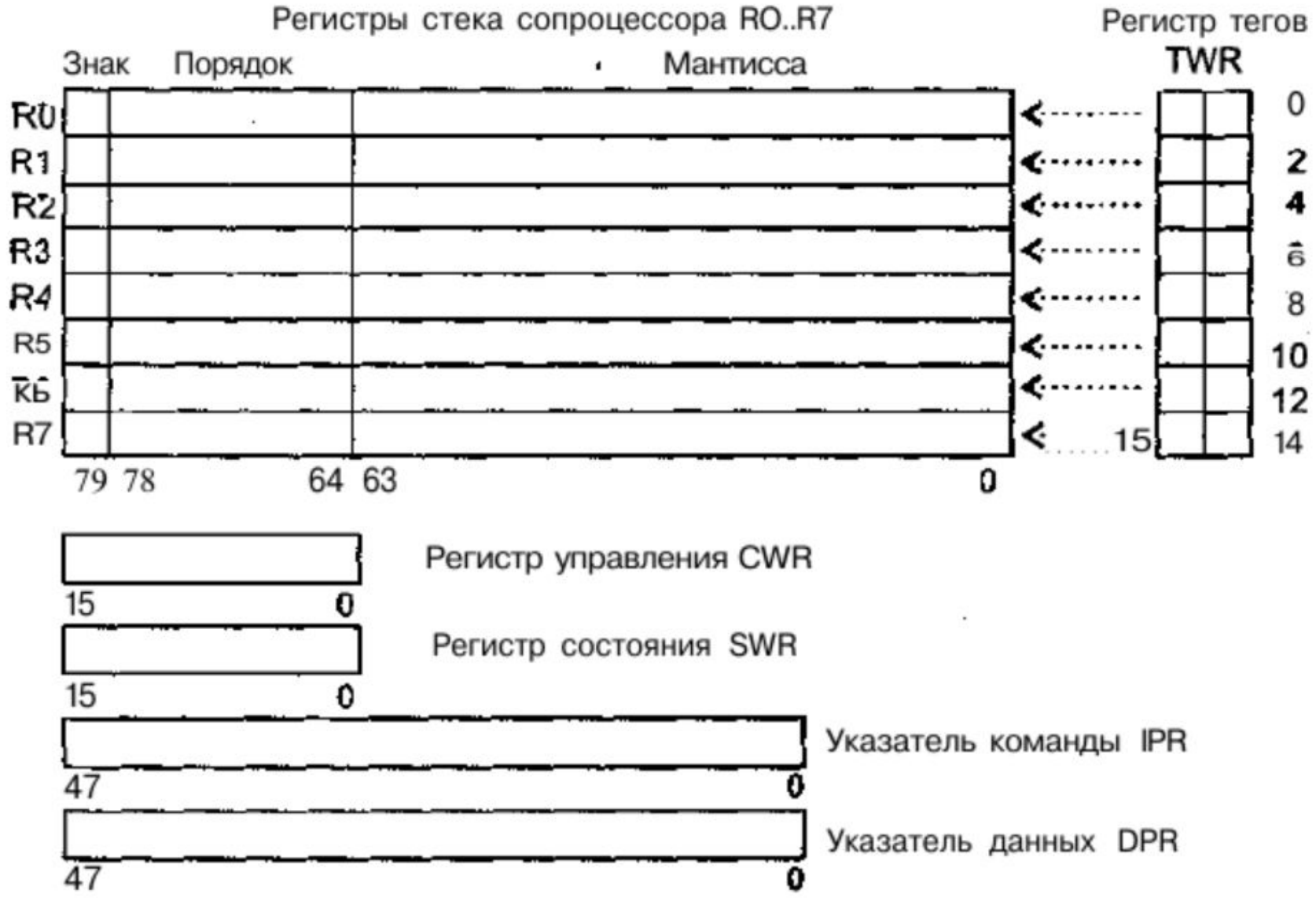
Неподдерживаемое число.

- Это все числа, которые не являются нормализованными числами или специальными значениями.

АРХИТЕКТУРА СОПРОЦЕССОРА

- **Стек регистров сопроцессора.** Регистры R0..R7 – предназначены для хранения вещественных операндов. Каждый регистр содержит 80 бит (0-63 – мантисса, 64-78 – порядок, 79 – знак числа). Оптимизированы на реализацию вычислений с использованием обратной польской записи.
- **Служебные регистры SWR, CWR и TWR** длиной 16 бит каждый.
  - SWR – регистр состояния сопроцессора. Содержит информацию о текущем состоянии сопроцессора, указывает, какой из регистров R0..R7 является вершиной стека сопроцессора, какие исключения возникли после выполнения последней команды и каковы особенности ее выполнения. Аналог регистра флагов центрального процессора.
  - CWR – управляющий регистр сопроцессора. С помощью его полей можно регулировать точность выполнения вычислений, управлять округлением, маскировать исключения.
  - TWR – регистр слова тегов. Используется для контроля за состоянием каждого из регистров R0..R7. Каждому из регистров стека сопроцессора в регистре TWR отведено по 2 бита: 0, 1 – R0; 2, 3 – R1 и т.д.
- **Регистры указателей DPR и IPR** длиной по 48 бит каждый.
  - Используются при обработке исключительных ситуаций.
  - DPR – регистр указателя данных. Хранит адрес операнда команды, вызвавшей исключение.
  - IPR – регистр указателя команды. Хранит адрес команды, вызвавшей исключение.

АРХИТЕКТУРА СОПРОЦЕССОРА



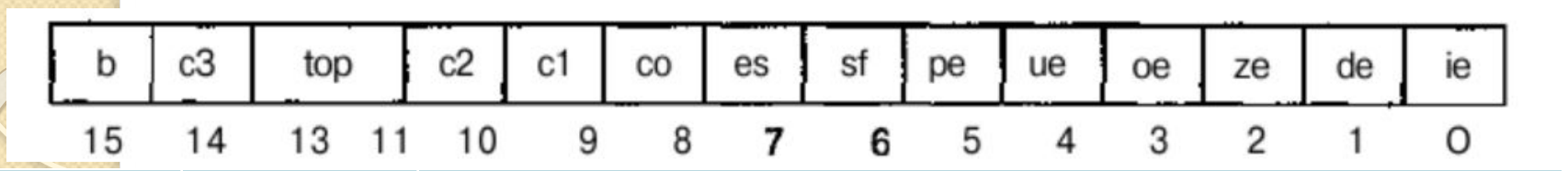
СТЕК СОПРОЦЕССОРА

- Физические регистры R0..R7.
- Размерность регистра – 80 бит.
- Тип данных – расширенный вещественный формат.
- Организован по принципу кольца.
- Вершина стека является плавающей и перемещается после записи операнда в вершину.
- Команды сопроцессора оперируют логическими номерами регистров, относительно вершины: ST(0), ST(1)...ST(7).
- ST(0)- вершина стека.

Каждому регистру R0..R7 соответствуют 2 бита регистра тегов TWR, характеризующие его состояние:

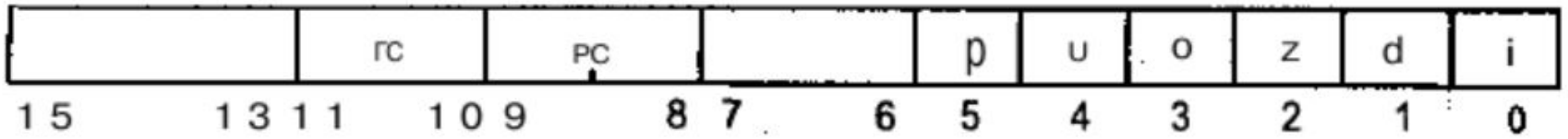
- 00 – регистр занят допустимым ненулевым значением,
- 01 – регистр содержит ноль,
- 10 – регистр содержит одно из специальных значений, кроме нуля,
- 11 – регистр пуст и в него можно записать число.

РЕГИСТР СОСТОЯНИЯ SWR



Бит	Обозначение	Назначение
0	IE	Недействительная операция
1	DE	Денормализованный операнд
2	ZE	Ошибка деления на нуль
3	OE	Ошибка переполнения – выход порядка за максимально допустимый диапазон значений
4	UE	Ошибка антипереполнения (результат слишком маленький)
5	PE	Ошибка точности – округление числа при выходе за пределы разрядной сетки
6	SF	Ошибка работы стека сопроцессора. 1 – возникла одна из исключительных ситуаций PE, UE или IE, выполнена попытка записи в заполненный стек или чтения из пустого стека.
7	ES	Суммарная ошибка работы сопроцессора. 1 – возникла любая из шести исключительных ситуаций (биты 0-5).
8	C_0	Код условия
9	C_1	Код условия
10	C_2	Код условия
11-13	TOP	Номер физического регистра R0..R7, который является текущей вершиной стека
14	C_3	Код условия
15	B	Бит занятости. 1 – сопроцессор выполняет команду или происходит прерывание от основного процессора. 0 – сопроцессор свободен

РЕГИСТР УПРАВЛЕНИЯ CWR



Бит	Обозначение	Назначение
0	I	Маски исключений. Предназначены для маскирования исключительных ситуаций, возникновение которых фиксируется битами 0-5 регистра SWR. 1 – соответст-вующее исключение обрабатывается самим сопроцессором. 0 – при возникновении исключения возбуждается прерывание 10h, обработчик которого должен быть написан программистом.
1	D	
2	Z	
3	O	
4	U	
5	P	
6	Зарезервировано	
7	IEM	Маска разрешения прерываний. 1 – даже при возникновении незамаскированного исключения (бит 0 -5 равен 0) прерывание не возбуждается.
8-9	PC	Поле управления точностью: 00 –мантисса занимает 24 бита, 10 – мантисса занимает 53 бита, 11 – мантисса занимает 64 бита.
10-11	RC	Поле управления округлением: 00 – округление по обычным правилам, 01 – округление в меньшую сторону, 10 – округление в большую сторону, 11 – отбрасывание дробной части результата (используется в операциях целочисленной арифметики).
13-15	Зарезервировано	

ВЗАИМОДЕЙСТВИЕ ЦП И СОПРОЦЕССОРА

- ЦП и сопроцессор работают параллельно.
- Очередная команда поступает одновременно и в ЦП и в сопроцессор.
- Если команда требует данных, ЦП извлекает их и выставляет на шину.
- Далее ЦП начинает декодировать следующую команду.
- Если команда требует данных, сопроцессор обращается к шине, получает данные и начинает выполнять команду.
- Необходима синхронизация ЦП и сопроцессора, т.к. ЦП быстрее обрабатывает команды сопроцессора.
- До процессоров 486 синхронизация выполнялась вручную программистом командами WAIT/FWAIT.
- Начиная с модели 486 команда WAIT/FWAIT введена в большинство команд сопроцессора, что обеспечивает его синхронизацию с ЦП.

ПОСТРОЕНИЕ ОБРАТНОЙ ПОЛЬСКОЙ ЗАПИСИ

Рассматриваем поочередно каждый символ:
1. Если этот символ - операнд, то помещаем его в выходную строку.
2. Если символ - знак операции (+, -, *, /), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет. Операции сложения и вычитания имеют меньший приоритет. Наименьший приоритет имеет открывающая скобка.
Получив один из этих символов, мы должны проверить стек:

а) Если стек пуст, или находящиеся в нем символы имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.
б) Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).

3. Если текущий символ - открывающая скобка, то помещаем ее в стек.
4. Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую, которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

ВЫЧИСЛЕНИЕ ОБРАТНОЙ ПОЛЬСКОЙ ЗАПИСИ

Пример.
Выражение: (a+b)*(c+d)-e.
ОПЗ: ab+cd+*e-.

Алгоритм вычисления:
1. Если очередной символ входной строки - операнд, то помещаем его в вершину стека.
2. Если очередной символ - знак операции, то извлекаем из стека два верхних операнда, выполняем над ними операцию, результат помещаем в вершину стека.
Когда вся входная строка будет разобрана в стеке должно остаться одно число, которое и будет результатом данного выражения.

Стек сопроцессора оптимизирован именно под этот алгоритм!

Работа с прерываниями, защищенный режим

ПОНЯТИЕ ПРЕРЫВАНИЯ

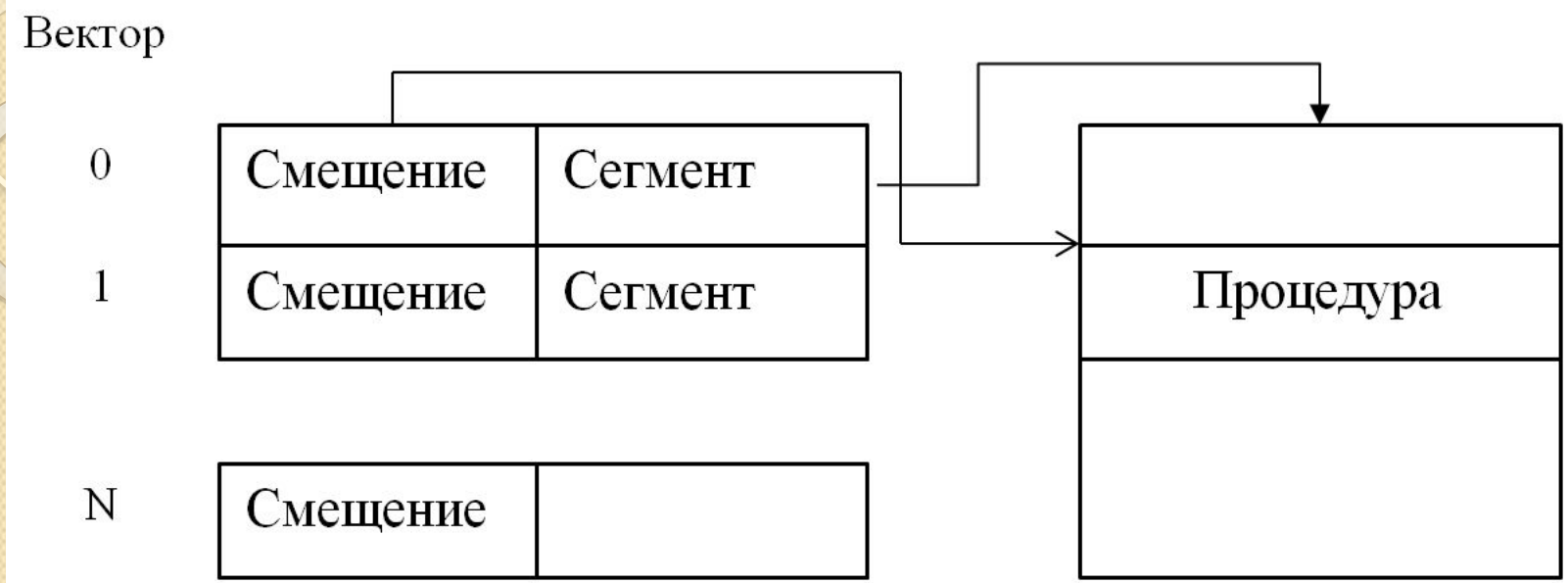
Прерывание– это временное прекращение некоторого программного блока с передачей управления другому программному блоку.

Прерывания разделяются на несколько видов:

- Программные – инициируются программным путем;
- Аппаратные внешние – инициируются внешними устройствами (клавиатура, мышь и т.д.);
- Аппаратные внутренние – инициируются внутри процессора (таймер, деление на о и т.д.).

- Обработка прерываний выполняется при помощи специальных системных подпрограмм, адреса которых записываются в таблицу векторов прерываний.
- В реальном режиме таблица векторов прерываний располагается по адресу 0 и содержит 256 векторов по 4 байта в каждом.
- Вектор содержит адрес сегмента (старшее слово) и смещения процедуры-обработчика в этом сегменте (младшее слово).

Вызов и возврат из прерывания



int НомерПрерывания – вызов прерывания

- В стек текущей программы заносится содержимое регистра флагов, сегментного регистра CS и указателя команд IP
- Номер прерывания совпадает с номером вектора, который расположен по адресу 0 + НомерВектора * 4.
- Программа может передать параметры обработчику прерывания в РОН.

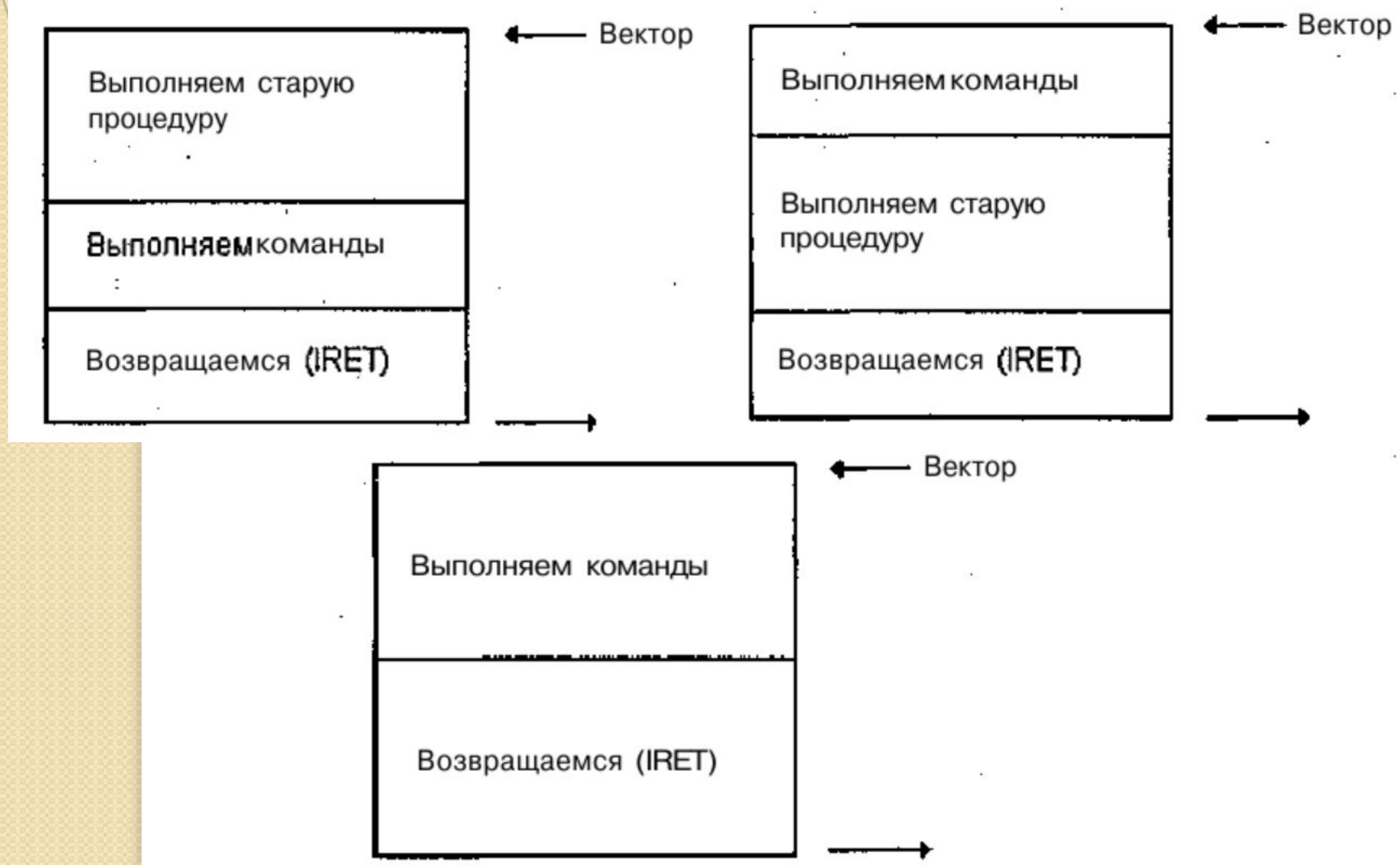
iret – возврат из прерывания

- Эта команда последовательно извлекает из стека значения регистров IP, CS (адрес возврата) и регистр флагов.
- Обработчик прерывания может возвращать в РОН некоторые значения и устанавливать флаги (изменив их в копии флагов в стеке).

ПЕРЕНАПРАВЛЕНИЕ ВЕКТОРА ПРЕРЫВАНИЯ

В некоторых случаях требуется изменить системный обработчик какого-либо прерывания или добавить к нему определенные действия.

Существует 3 варианта.



ПЕРЕНАПРАВЛЕНИЕ ВЕКТОРА ПРЕРЫВАНИЯ. Способ 1

Используем функции DOS (прерывание 21h) с кодами 25h и 35h. Первая позволяет установить вектор на свою процедуру, а вторая – получить вектор.

Пример.

```
MOV AH, 35H                ;получить вектор прерывания
MOV AL, 5                  ;вектор 5 (печать экрана) :
INT 21H ;после выполнения содержимое вектора в ES:BX
MOV OLD_S, ES              ;сохранить старый вектор (сегмент)
MOV OLD_0, BX              ; (смещение)
MOV AH, 25H                ; установить вектор на свою процедуру
MOV DX, ProcOffset         ;смещение
MOV CX, ProcSegment        ;сегмент
MOV DS, CX
INT 21H
```

ПЕРЕНАПРАВЛЕНИЕ ВЕКТОРА ПРЕРЫВАНИЯ. Способ 2

Непосредственное занесение значения в вектор.

Пример.

```
CLI                    ;запретить обработку прерываний
MOV AX, 0
MOV ES, AX             ; адрес сегмента - 0
MOV DX, ES:[5H*4]      ; смещение процедуры вектора в DX
MOV BX, ES: [5H*4+2]   ; сегмент в BX
MOV OLD_S, BX          ;сохраняем старый
MOV OLD_0, DX          ;вектор
MOV DX, ProcOffset     ; смещение устанавливаемой процедуры
MOV AX, ProcSegment    ; сегмент процедуры
MOV ES:[5H*4], DX      ;изменяем вектор
MOV ES:[5H*4+2],AX
STI                    ;разрешить прерывание
```

Здесь команды CLI и STI нужны для запрета вызова именно этого прерывания.

Вызов стандартного обработчика прерывания

Способ 1. Используем команду CALL.

Пример. Пусть O_INT – смещение (младшее слово), а S_INT – сегмент (старшее слово), расположенные в сегменте данных. Тогда переход выполняется:

```
PUSHF          ; сохраняем регистр флагов для правильного возврата
               ; командой iRet
CALL DWORD PTR DS:[O_INT]     ; косвенный вызов процедуры
```

Здесь нужно быть очень аккуратным, поскольку эта процедура может возвращать результат либо в регистрах, либо во флагах. Нужно обеспечить, чтобы эти значения регистров и флагов не изменялись.

Способ 2. Для вызова прерывания можно также использовать свободные векторы. Для этого необходимо направить неиспользуемый вектор на нужную процедуру и затем вызвать его командой INT.

Пример. Мы работаем с вектором 16H. Он направлен на нашу процедуру. Старое значение вектора 16H присваиваем, например, вектору FEH. В конце процедуры обработки ставим команду

```
INT FEH.
```

Такой вызов будет аналогичен вызову через CALL.

Внимание! Если программа изменила вектор прерывания, то перед ее завершением необходимо восстановить его старое значение!!!

ЗАЩИЩЕННЫЙ РЕЖИМ

Защищенный режим позволяет использовать дополнительные возможности процессоров:

- увеличение адресуемого пространства до 4 Гбайт;
- возможность работать в виртуальном адресном пространстве, превышающем максимально возможный объем физической памяти и достигающем 64 Тбайт;
- организация многозадачного режима с параллельным выполнением нескольких программ (процессов);
- страничная организация памяти, повышающая уровень защиты задач друг от друга и эффективность их выполнения.

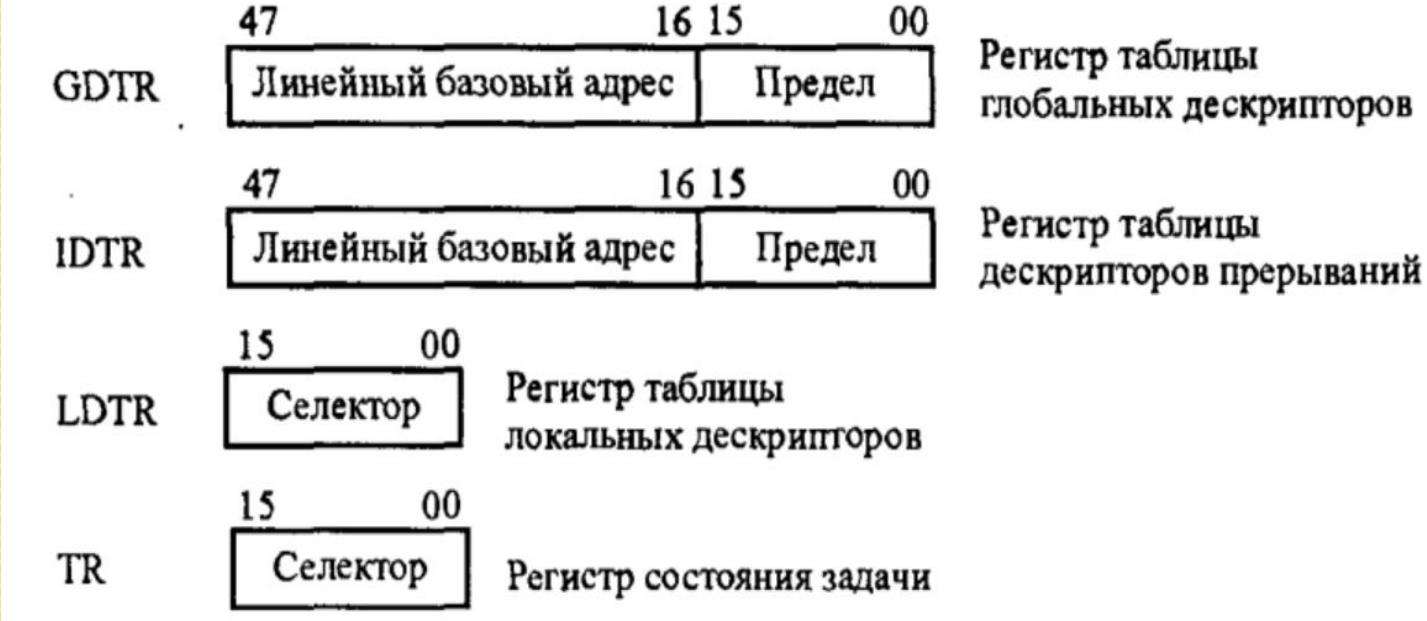
В 32-разрядных процессорах появились 4 управляющих регистра CR0..CR3, в которых содержится информация о состоянии процессора. Регистры доступны только в защищенном режиме.

- CR0 –слово состояния системы, биты которого задают режимы работы:
  - Бит разрешения защиты **PE** (бит 0). **PE**=1 – процессор работает в защищенном режиме, **PE**=0 – в реальном режиме.
  - Бит страничного преобразования **PG** (бит 31). **PG**=1 – страничное преобразование включено, **PG**=0 – выключено.
- CR1 – зарезервирован, CR2 и CR3 используются для страничного преобразования адреса.
- Доступ к этим регистрам имеет программа с наивысшим уровнем привилегий (0). Меняя бит **PE** можно переключаться в защищенный режим и обратно. Если же программа не имеет привилегий, переключение в защищенный режим выполняется при помощи системных функций, вызываемых через прерывания.

РЕГИСТРЫ СИСТЕМНЫХ АДРЕСОВ

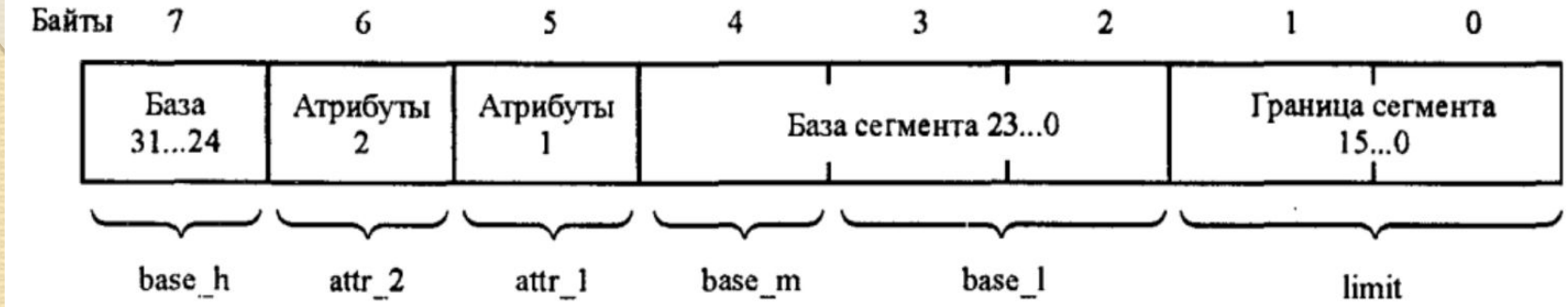
В состав процессора входят 4 регистра системных адресов:

- GDTR (Global Descriptor Table Register) - регистр таблицы глобальных дескрипторов;
- LDTR (Local Descriptor Table Register) - регистр таблицы локальных дескрипторов;
- IDTR (Interrupt Descriptor Table Register) - регистр таблицы дескрипторов прерываний;
- TR (Task Register) - регистр состояния задачи для хранения селектора сегмента состояния задачи.



ДЕСКРИПТОРЫ

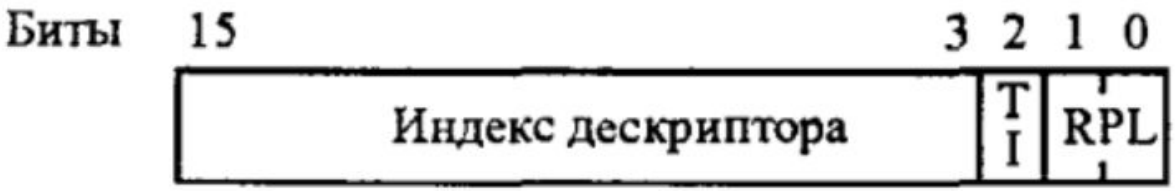
- В защищенном режиме для каждого сегмента программы должен быть определен дескриптор – 8-байтовое поле, в котором записываются базовый адрес сегмента и его длина.



- База сегмента (32 бита) определяет начальный линейный адрес сегмента в адресном пространстве процессора.
- Граница (limit) сегмента представляет собой номер последнего байта сегмента.
- Граница может указываться либо в байтах (тогда максимальный размер сегмента равен 1 Мбайт), либо в блоках по 4 Кбайт (тогда размер сегмента может достигать 4 Гбайт, но будет кратен 4 К).
- В каких единицах задастся граница - определяет специальный бит дробности в атрибутах дескриптора.
- Дескрипторы размещаются либо в таблице глобальных дескрипторов GDT либо в таблице локальных дескрипторов LDT. Таблица GDT может быть только одна, а таблиц LDT может быть произвольное количество. Сегменты GDT доступны всем задачам, а сегменты LDT – только в пределах своей задачи.

СЕЛЕКТОР ДЕСКРИПТОРА

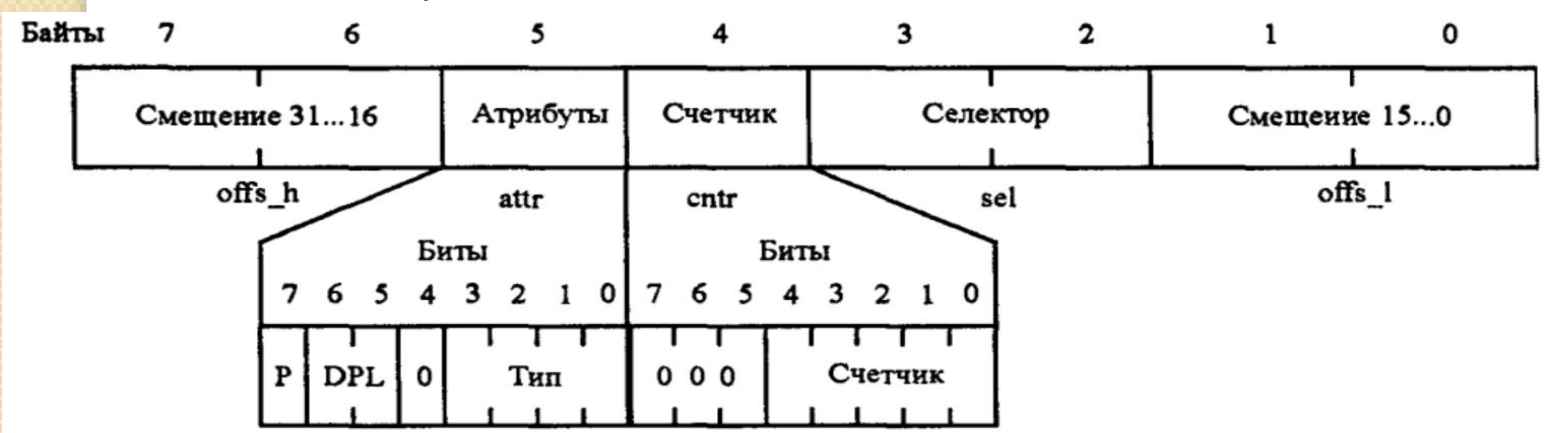
- Для обращения к требуемому сегменту программист заносит в сегментный регистр не сегментный адрес, а так называемый селектор.
- В состав селектора входит номер (индекс) соответствующего сегменту дескриптора.
- Процессор по этому номеру находит нужный дескриптор, извлекает из него базовый адрес сегмента и, прибавляя к нему указанное в конкретной команде смещение (относительный адрес), формирует адрес ячейки памяти.
- Индекс дескриптора записывается в селектор начиная с бита 3, что эквивалентно умножению его на 8. Таким образом, можно считать, что селекторы последовательных дескрипторов представляют собой числа 0, 8, 16, 24 и т. д.



RPL – уровень привилегий приложения;
TI – задает таблицу дескрипторов (0-глобальная, 1-локальная).

ПРЕРЫВАНИЯ В ЗАЩИЩЕННОМ РЕЖИМЕ

- В защищенном режиме аналогом таблицы векторов прерываний является таблица дескрипторов прерываний - IDT (Interrupt Descriptor Table), располагающаяся обычно в операционной системе защищенного режима.
- Таблица IDT содержит дескрипторы обработчиков прерываний, в которые входят их адреса.
- Для того чтобы процессор мог обратиться к этой таблице, ее адрес следует загрузить в регистр IDTR (Interrupt Descriptor Table Register, регистр таблицы дескрипторов прерываний).
- Таблица дескрипторов прерываний IDT состоит из дескрипторов, которые называются шлюзами. Через шлюзы осуществляется доступ к обработчикам прерываний и исключений.
- Формат шлюза отличается от формата дескриптора сегмента памяти.
- Основной частью шлюза является полный трехсловный адрес обработчика, состоящий из селектора и смещения.



ПЕРЕХОД В ЗАЩИЩЕННЫЙ РЕЖИМ

Для перехода в защищенный режим, нужно:

- Создать и заполнить таблицы глобальных (GDT) и локальных (LDT) дескрипторов.
- Сформировать таблицу дескрипторов прерываний IDT.
- Загрузить адреса этих таблиц в регистры GDTR, LDTR, IDTR.
- Загрузить в сегментные регистры селекторы дескрипторов.
- Перейти в защищенный режим.

Для перехода в защищенный режим из приложения реального режима предназначены специальные системные процедуры, реализованные в виде интерфейсов VCPI и DPMI.