Архитектура младшей модели семейства Intel

Память

- Рассматриваемый компьютер имеет архитектуру с адресуемыми регистрами, адресуемая память состоит из основной и регистровой памяти. Основная память имеет объём 2^{20} ячеек по 8 бит каждая, при этом каждая команда или данные располагаются в одной или нескольких последовательных (с возрастающими адресами) ячейках этой памяти.

Форматы данных Вещественные числа

- На современных ЭВМ чаще всего используются три формата вещественных чисел:
  - короткие (длиной 4 байта),
  - длинные (8 байт)
  - и сверхдлинные (16 байт) вещественные числа.
- На момент *массового* выпуска ЭВМ с командами для работы с вещественными числами, уже существовал международный стандарт на внутреннее представление этих чисел (ANSI/IEEE standart 754-1985), и почти все современные машины придерживаются этого стандарта на представление вещественных чисел.

Целые числа

- Целые числа в младшей модели могут занимать в памяти 8 бит (короткое целое), 16 бит (длинное целое) и 32 бита (сверхдлинное целое). Длинное целое принято называть *машинным словом*

(не путать с машинным словом в машине Фон Неймана, там это содержимое одной ячейки памяти!).

Символьные данные

- В качестве символов используются короткие целые числа, которые трактуются как неотрицательные (беззнаковые) числа, задающие номер символа в некотором алфавите.

Массивы (строки)

- Допускаются только одномерные массивы, которые могут состоять из коротких или длинных целых чисел. Массив коротких целых чисел может рассматриваться программистом как *символьная строка*, отсюда и второе название этой структуры данных. В машинном языке присутствуют команды для обработки *элементов* таких массивов

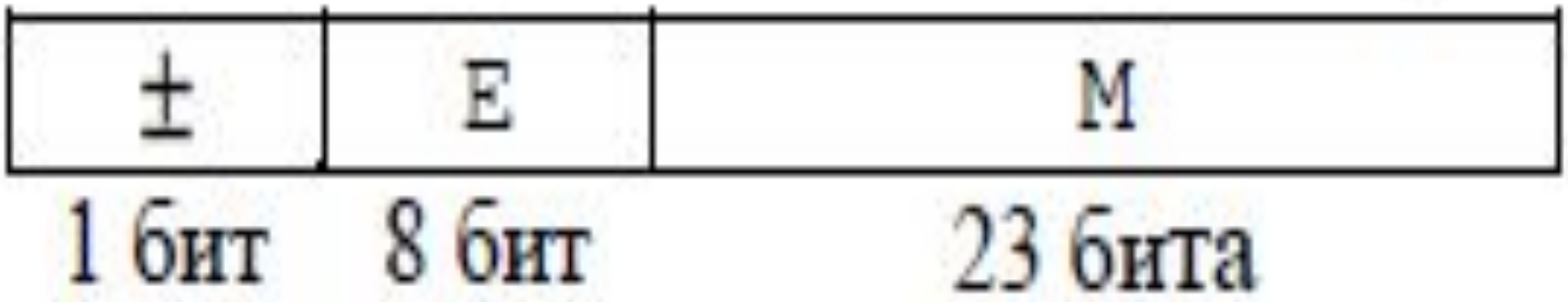
- **Логические (битовые) вектора.**

В языке машины представлены команды для обработки логический векторов длиной 8 или 16 бит. Элементы таких векторов трактуются как логические переменные..

- **Двоично-десятичные целые числа.**

Это целые числа в двоично-десятичной записи, имеющие размер до 16 байт.

Вещественные числа



Вещественные числа

$$A = \pm 1.M * 2^{E-127}.$$

Такое представление вещественного числа называется *нормализованным*:

его первый сомножитель удовлетворяет неравенству:

$$1.0 \leq 1.M < 2.0$$

–13.25

- Сначала переведём его в двоичную систему счисления:

$-13.25_{10} = -1101.01_2$

- Затем нормализуем это число:

$-1101.01_2 = -1.10101_2 * 2^3$

- Следовательно, мантисса нашего числа будет иметь вид $10101000000000000000000_2$, и осталось вычислить машинный порядок

E: 3 = E-127; E = 130 = 128+2 = 10000010_2 .

- Учитывая знак, получаем вид внутреннего машинного представления числа $–13.25_{10}$:

$1100\ 0001\ 0101\ 0100\ 0000\ 0000\ 0000\ 0000_2 = C1540000_{16}$

Шестнадцатеричные числа в языке Ассемблера принято записывать с буквой h на конце,

- при этом, если такое число начинается с буквы, то впереди записывается незначащий ноль, чтобы отличить запись такого числа от *имени*:
- C154000016 = 0C1540000h

Представимый диапазон порядков коротких вещественных чисел равен

$$2^{-126}..2^{127} \approx 10^{-38}..10^{38}$$

Из-за конечной длины представления вещественных чисел действия с ними выдают приближённый результат

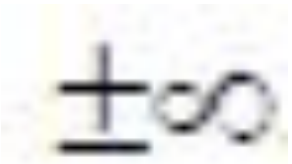
Возможно:

- Возможны случаи, когда (a+b)+c ≠ a+(b+c) и (a+b)*c ≠ a*c+b*c.
- Решение простейшего уравнения X+A=A будет равен, скажем, 10^{+6}.

Некоторые комбинации нулей и единиц в памяти, отведённой под хранения вещественного числа, используются для служебных целей.

- Значение машинного порядка E=255 при мантиссе M ≠ 0 обозначает специальное значение "*не число*" (NAN – not a number).

- Машинный порядок E=255 при мантиссе M = 0 задаёт, в зависимости от знака числа, специальные значения



Целые числа

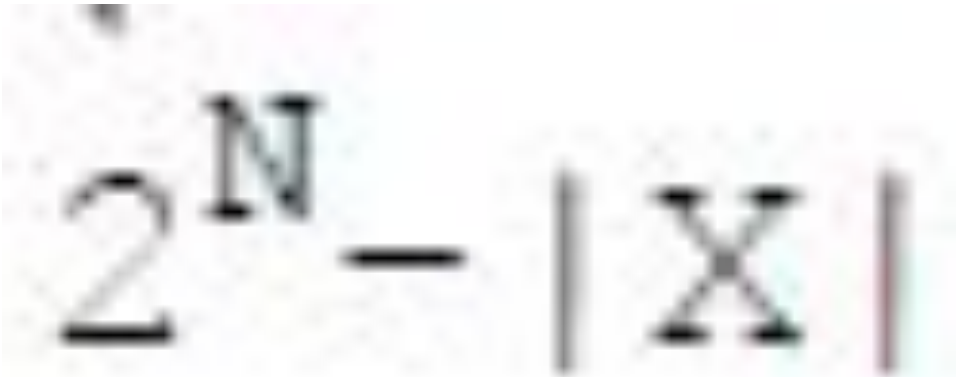
- Беззнаковые (неотрицательные) числа представляются в двоичной системе счисления - *прямым кодом*
- Если *инвертировать* прямой код (т.е. заменить все "1" на "0", а все "0" на "1"), то получим *обратный* код числа.
- Для представления *отрицательных* знаковых чисел используется *дополнительный* (complementary) код, который можно получить из обратного кода прибавлением единицы.

Прямой код	00001101
Обратный код	11110010
+	1
Дополнительный код	11110011

Другой способ

Дополнительный код числа –13 можно вычислить и так:

- 2^8-13 = 256–13 = 100000000–00001101 = 11110011



Если сложить дополнительный код с прямым кодом, то получится ноль и "лишняя" единица, не помещающаяся в отводимое число разрядов.

Результат операций могут быть разными для знаковых и беззнаковых чисел

Пример 1.

	Б/з.	Знак.
11111100	252	-4
00000101	5	5
100000001	1	1

Флаги

Для таких ситуаций в архитектуре компьютера введено понятие *флагов*. Каждый флаг занимает один бит в специальном *регистре флагов* с именем FLAGS. Для рассмотренного выше примера флаг CF (carry flag) после сложения примет значение, равное единице (иногда говорят, что флаг *поднят*), сигнализируя программисту о том, что при без-знаковом сложении произошла ошибка. Рассматривая результат в знаковых числах, мы получили *правильный* ответ, поэтому флаг результата знакового сложения OF (overflow flag) будет положен равным нулю (или, как говорят, *опущен*).

- Флаг CF называется *флагом переноса*,
- а OF – *флагом переполнения*.

- Существует флаг SF, в который всегда заносится знаковый (крайний левый) бит результата, таким образом, при знаковой трактовке чисел этот флаг сигнализирует, что результат получился отрицательным.
- Флаг ZF, устанавливается в 1, если результат тождественно равен нулю, в противном случае этот флаг устанавливается в 0. Флаги в нашей архитектуре выполняют ту же роль, что и регистр признака результата ω в изученной ранее учебной ЭВМ УМ-3.

Сегментация памяти

- Память нашей ЭВМ имеет сегментную организацию. В любой момент времени для младшей модели определены четыре сегмента (хотя для старших моделей число сегментов больше). Есть четыре *сегментных* регистра, которые указывают на определённые области памяти. Каждый сегментный регистр имеет длину 16 разрядов, а в то же время для адресации любого места нашей памяти необходимо, как мы уже говорили, 20 разрядов. Для того чтобы сегмент мог указывать на некоторое место оперативной памяти, адрес начала сегмента получается после умножения значения сегментного регистра на число 16. При таком способе задания начала сегмента, он может начинаться не с любого места оперативной памяти, а только с адресов, кратных 16 (в некоторых книгах по Ассемблеру такие участки памяти называются *параграфами*).

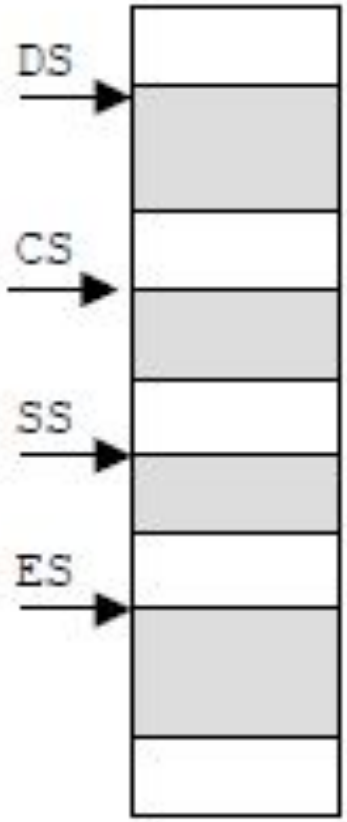
Физический адрес числа или команды вычисляется центральным процессором по формуле

$$A_{физ} := (SEG*16 + A) \bmod 2^{20}.$$

В качестве мнемонических обозначений сегментных регистров выбраны следующие двухбуквенные *служебные* имена:

- кодовый сегментный регистр (CS),
- сегментный регистр данных (DS),
- сегментный регистр стека (SS)
- дополнительный сегментный регистр (ES).

- Сегментные регистры являются специализированными, предназначенными только для хранения адресов сегментов, поэтому арифметические операции (сложение, вычитание и др.) над их содержимым в языке машины не предусмотрены.
- Заметим, что даже если все сегменты не перекрываются и имеют максимальный размер, то и в этом случае центральный процессор в каждый момент времени имеет доступ только к одной четвёртой от общего объёма оперативной памяти.



Пример положения сегментов в памяти.

только для хранения адресов сегментов

СЕГМЕНТНЫЕ РЕГИСТРЫ		
CS	Code Segment	Сегмент команд, не может быть изменен напрямую
SS	Stack Segment	Сегмент стека
DS	Data Segment	Сегмент данных
ES	Extension Segment	Дополнительный сегмент

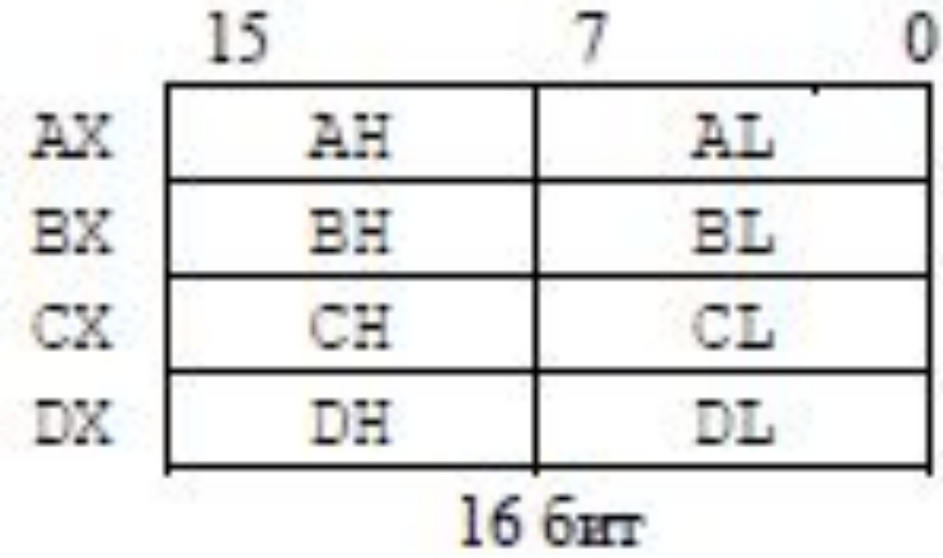
Мнемонические обозначения регистров

Регистры общего назначения, каждый из которых может складывать, вычитать и просто хранить данные, а некоторые – ещё умножать и делить, обозначают следующими служебными именами:

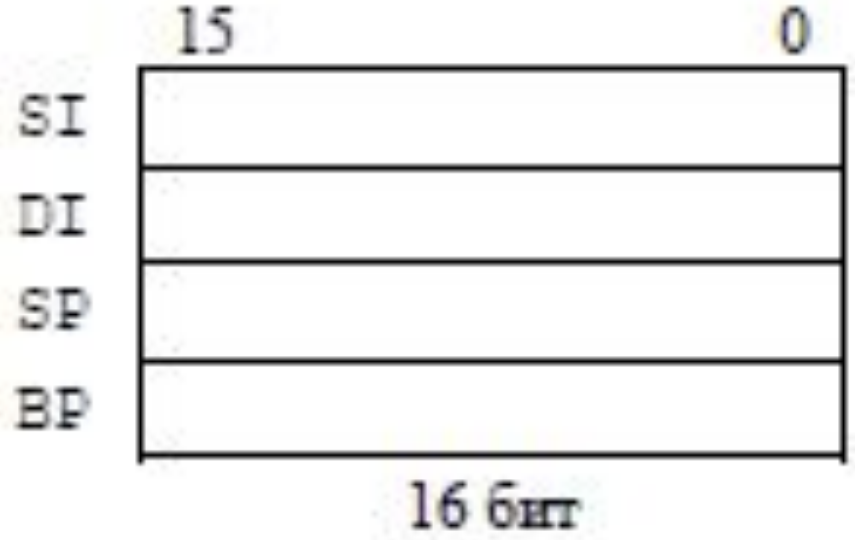
- AX, BX, CX, DX.

Для обеспечения многообразия форматов данных каждый из них разбит на две части по 8 бит.

Каждый из регистров AH, AL, BH, BL, CH, CL, DH и DL может быть использован в машинных командах как самостоятельный регистр, на них можно выполнять операции сложения и вычитания.



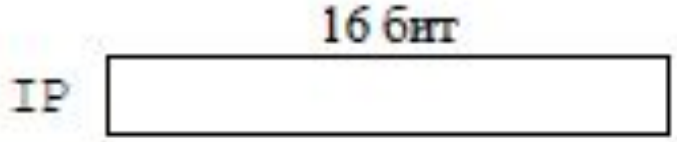
Существуют также четыре регистра с именами SI, DI, SP и BP, которые также могут использоваться для проведения сложения и вычитания, но они уже не делятся на половинки:



•В основном эти четыре регистра используются как *индексные*, т.е. на них обычно храниться положение конкретного элемента в некотором массиве. Условное обозначение - r16 (а AH, AL, BH, BL, CH, CLи т.п. – r8)

РЕГИСТРЫ ОБЩЕГО НАЗНАЧЕНИЯ			
AX	Accumulator	Основной сумматор	AH,AL
BX	Base	Адресация по базе	BH,BL
CX	Counter	Счетчик циклов	CH,CL
DX	Data	Для "длинных" данных	DH,DL
SI	Source Index	Индексирование источника	
DI	Destination Index	Индексирование приемника	
BP	Base Pointer	База стека	
SP	Stack Pointer	Вершина стека	

- Кроме перечисленных выше регистров программист имеет дело с регистром IP (instruction pointer), который называется счётчиком адреса (в учебной машине мы обозначали его как RA). Этот регистр содержит адрес следующей исполняемой команды (точнее, содержит *смещение* этой команды относительно начала кодового сегмента, адрес начала этого сегмента равен значению сегментного регистра CS, умноженному на 16).



- И, наконец, как уже упоминалось, архитектурой изучаемой ЭВМ предусмотрен регистр флагов с именем FLAGS. Он содержит шестнадцать одноразрядных флагов, например, флаги CF и OF.

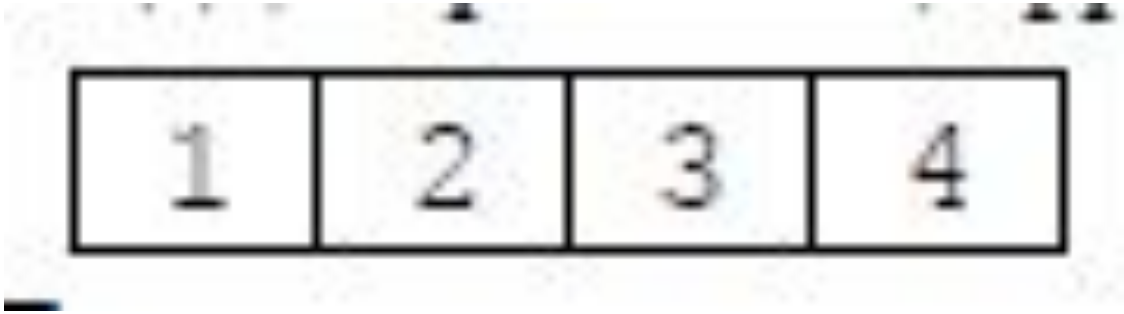
РЕГИСТР ФЛАГОВ			
FLAGS	Flags	Информация о текущем состоянии процессора	
РЕГИСТР УКАЗАТЕЛЯ КОМАНД			
IP	Instruction Pointer	Смещение команды, программно недоступен	

Биты регистра флагов перечислены в следующей таблице:

Бит	Имя	Название	Тип	Назначение
0	CF	Carry Flag	сост.	Перенос или заем
2	PF	Parity Flag	сост.	Четность
4	AF	AuxiliaryFlag	сост.	Перенос или заем BDC
6	ZF	Zero Flag	сост.	Ноль результата
7	SF	Sign Flag	сост.	Знак результата
8	TF	Trace Flag	упр.	Трассировка
9	IF	Interrupt Flag	упр.	Разрешение прерываний
10	DF	DirectionFlag	упр.	Напр. обработки цепочек
11	OF	Overflow Flag	сост.	Переполнение

Особенности хранения чисел в регистровой и основной памяти ЭВМ

- Запишем, например, шестнадцатеричное число 1234h в какой-нибудь 16-тиразрядный регистр (каждая шестнадцатеричная цифра занимает по 4 бита):



- Перешлём машинной командой содержимое этого регистра в память в ячейки с адресами, например, 100 и 101. В ячейку с первым (старшим) адресом 100 при такой пересылке запишется число из младшего байта регистра 34h, а в ячейку со вторым (младшим) адресом 101 запишется число из первого (старшего) байта регистр 12h.
- Говорят, что целое число представлено в основной памяти (в отличие от регистров) в *перевёрнутом* виде.

Структура команд

Первое поле команды – код операции – занимает 6 первых бит(до 64 различных операций).
Однобитные поля с именами d и w,
где d –бит *направления*, а w – бит *размера аргумента*,
последующие два бита для этого формата всегда равны 11,
а последние две части (по 3 бита каждая) -

Формат регистр–регистр.

6 бит	1 бит	1 бит	2 бита	3 бита	3 бита
КОП	d	w	11	R1	R2

Команды этого формата занимают в памяти 2 байта.
Первое поле команды – код операции – занимает 6 первых бит(до 64 различных операций).
Однобитные поля с именами d и w,
где d –бит *направления*, а w – бит *размера аргумента*,
последующие два бита для этого формата всегда равны 11,
а последние две части (по 3 бита каждая) -
задают номера регистров-операндов команды.

Назначение битов d и w

- Бит d задаёт направление выполнения команды, а именно:

<R1> := <R1> ⊗ <R2> **при** d = 0
<R2> := <R2> ⊗ <R1> **при** d = 1.

Бит w задаёт размер регистров-операндов, а соответствие двоичных номеров регистров и их имён можно определить по таблице

$r_{1,2}$	w = 1	w = 0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

- Для проведения операций над числами разной длины появляется необходимость *преобразования типов* из короткого целого в длинное, и из длинного в сверхдлинное (и наоборот). Такое преобразование зависит от знаковой или беззнаковой трактовки числа.

Для преобразования знаковых целых чисел из более короткого формата в более длинный в языке машины предусмотрены безадресные команды, имеющие в Ассемблере такую мнемонику:

- **cbw** (convert byte to word)

и

- **cwd** (convert word to double),
- которые производят *знаковое* расширение соответственно значения регистра AL до AX и AX до значения пары регистров <DX,AX> (так называемой *регистровой пары*), которые в этом случае рассматриваются как один длинный 32-х битный регистр.

Формат регистр–память (и память-регистр).

- КОП r1 A2

Второй операнд A2 может в этом формате иметь один из приведённых ниже трёх видов:

1. A2 = A,
2. A2 = A[M1],
3. A2 = A[M1][M2].

- Здесь A – задаваемое в команде число (смещение) длиной 1 или 2 байта (заметим, что нулевое смещение иногда может не задаваться и совсем не занимать места в команде), M1 и M2 – так называемые *регистры-модификаторы*. Как мы сейчас увидим, значение адреса второго операнда A2 будет *вычисляться* по определённым правилам, поэтому этот адрес часто называют *исполнительным* адресом.

- Подробнее каждый их трёх возможных видов второго операнда A2 см. в Приложении «Возможные виды второго операнда»

Команды языка машины
Команды пересылки

Все они пересылают значение одного или двух байт из одного места памяти в другое. Для более компактного описания синтаксиса машинных команд введём следующие условные обозначения (с некоторыми из них мы уже знакомы):

r8 – любой короткий регистр AH,AL,BH,BL,CH,CL,DH,DL;

r16 – любой из длинных регистров AX,BX,CX,DX,SI,DI,SP,BP;

m8, m16, m32 – операнды, расположенные в основной памяти длиной 1,2 и 4 байта;

i8, i16, i32 – непосредственные операнды в самой команде длиной 1, 2 и 4 байта;

SR – один из трёх сегментных регистров SS, DS, ES;

CS – кодовый сегментный регистр.

Общий вид команды пересылки в двухадресной ЭВМ такой (после точки с запятой будем записывать, как это принято в Ассемблере, *комментарий* к команде):

mov op1,op2; op1 := op2

Существуют следующие допустимые форматы первого и второго операндов команды пересылки, запишем их в виде таблицы, где во второй колонке перечислены все возможные вторые операнды, допустимые для операнда из первой колонки:

op1	op2
r8	r8, m8, i8
r16	r16, m16, i16, SR, CS
m8	r8, i8
m16	r16, i16, SR, CS
SR	r16, m16

Команды пересылок не меняют флаги в регистре FLAGS. Команды пересылок с кодом операции **mov** бывают форматов

- RR, RX (и XR), RI и SI.
- Команда *обмена* содержимым двух операндов, команда также не меняет флаги
- **xchg** op1,op2;
- Таблица допустимых операндов для этой команды:

op1	op2
r8	r8, m8
m8	r8
r16	r16, m16
m16	r16

Арифметические команды

- **КОП** op1,op2,
- где **КОП** = **add**, **sub, adc**, **sbb**.
- Команды с кодами операций **add** (сложение) и **sub** (вычитание) выполняются по схеме:
- op1 := op1 ± op2

Команды с кодами операций **adc** (сложение с учётом флага переноса) и **sbb** (вычитание с учётом флага переноса) имеют *три* операнда, два из которых задаются в команде явно, а третий по
умолчанию является значением флага переноса CF:

op1 := op1 ± op2 ± CF

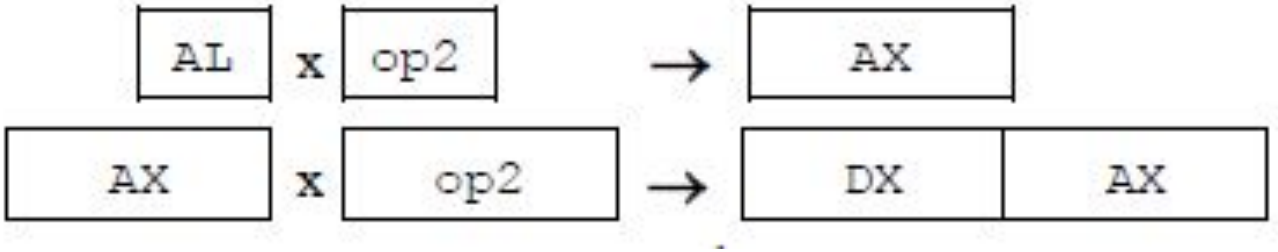
Таблица допустимых операндов для этих команд:

op1	op2
r8	r8, m8
m8	r8
r16	r16, m16
m16	r16

Команды умножения и деления целых чисел

- Первый операнд всех команд этого класса явно в команде не указывается и находится в фиксированном регистре, заданном *по умолчанию*. В младшей модели семейства есть следующие команды умножения и деления, в них, как и в одноадресной ЭВМ, явно задаётся только второй операнд (т.е. второй сомножитель или делитель):
- **mul** op2; беззнаковое умножение,
- **imul** op2; знаковое умножение,
- **div** op2; беззнаковое целочисленное деление,
- **idiv** op2; знаковое целочисленное деление.

- В случае с коротким вторым операндом форматов r8 и m8 при умножении вычисление производится по формуле:
- AX := AL * op2
- В случае с длинным вторым операндом форматов r16 и m16 при умножении вычисление производится по формуле:
- <DX,AX> := AX * op2
- Как видим, в этом случае произведение располагается сразу в двух регистрах <DX,AX> (это называется *регистровой парой*).
- Схема выполнения короткого и длинного умножения.



При делении на короткий операнд форматов r8 и m8 производятся следующие действия:

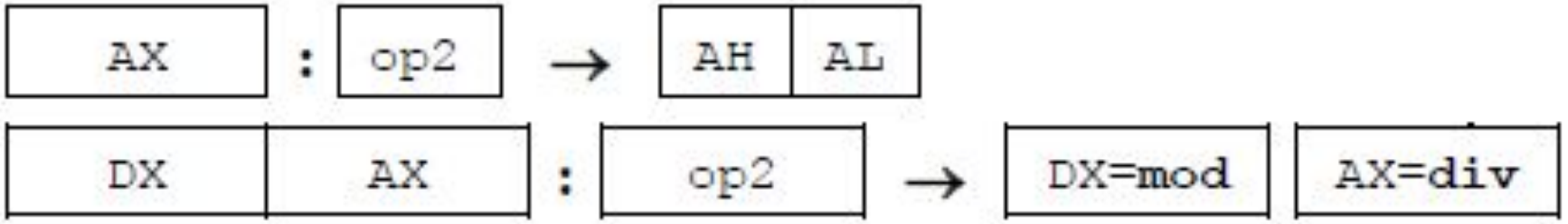
- AL := AX **div** op2
- AH := AX **mod** op2

При делении на длинный операнд формата r16 и m16 вычисление производится по формулам:

- AX := <DX,AX> **div** op2
- DX := <DX,AX> **mod** op2

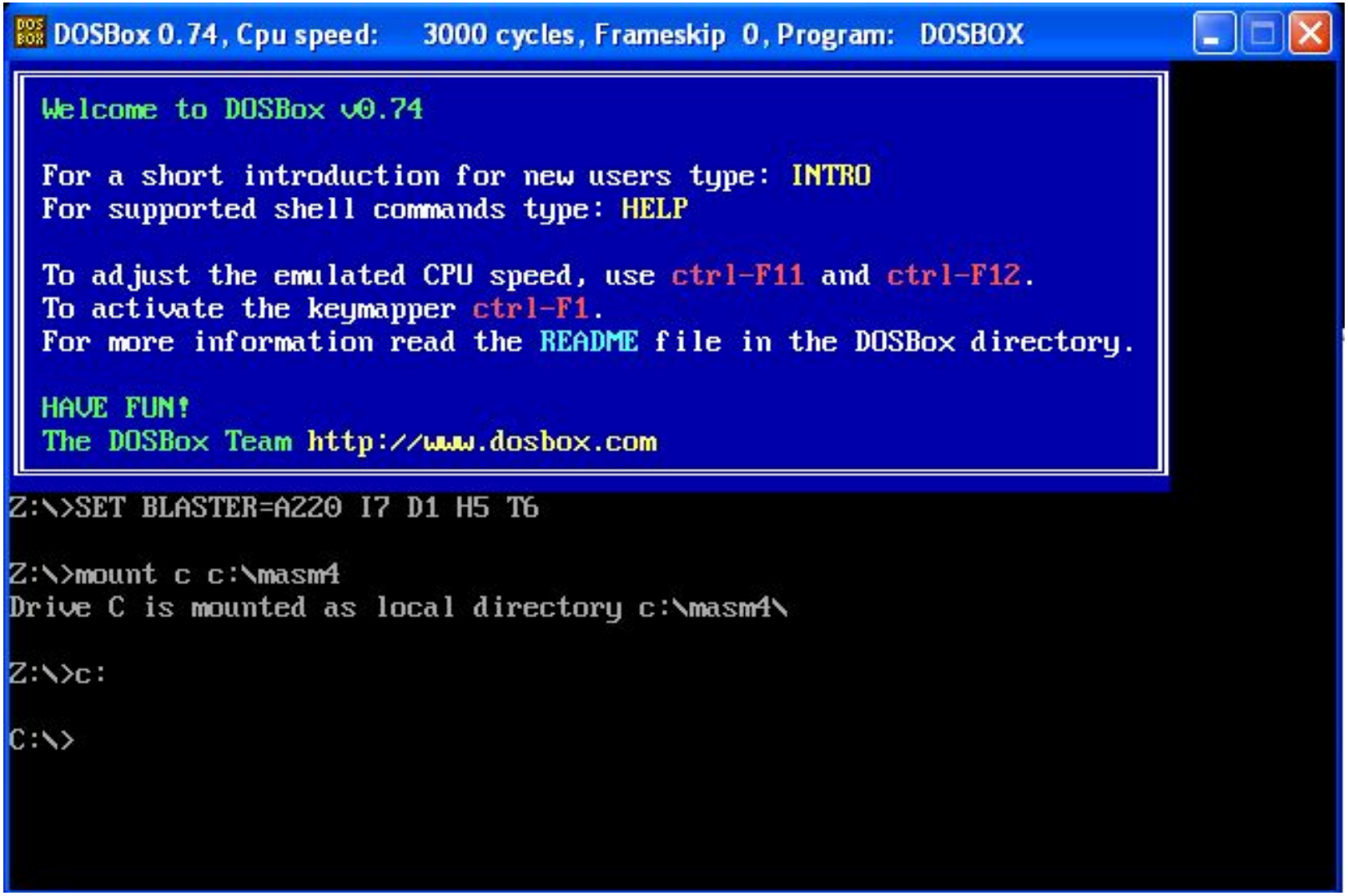
В этих командах операнд запись <DX,AX> обозначает 32-разрядное целое число, расположенное сразу в двух регистрах DX и AX, а op2, как уже говорилось, может иметь формат r16 или m16

Схема выполнения короткого и длинного деления



- После выполнения команд умножения устанавливаются некоторые флаги, из которых для программиста представляют интерес только флаги переполнения и переноса (CF и OF).
- Эти флаги устанавливаются по следующему правилу. CF=OF=1, если в произведении столько значащих (двоичных) цифр, что они не помещаются в *младшей* половине произведения. На практике это означает, что при значениях флагов CF=OF=1 произведение коротких целых чисел не помещается в регистр AL и частично "переползает" в регистр AH. Аналогично произведение длинных целых чисел – не помещается в регистре AX и "на самом деле" занимает оба регистра <DX,AX>. И наоборот, если CF=OF=0, то в старшей половине произведения (соответственно в регистрах AH и DX) находятся только *незначащие* двоичные цифры произведения (это двоичные нули для положительных и двоичные единицы для отрицательных произведений). Другими словами, при CF=OF=0 в качестве результата произведения можно взять только его младшую половину, что может оказаться полезным при программировании.

- Для написания программ на Ассемблере нам будут нужны также следующие *унарные* арифметические операции.
- **neg** op1; взятие обратной величины знакового числа, op1:=-op1;
- **inc** op1; увеличение (инкремент) аргумента на единицу, op1:=op1+1;
- **dec** op1; уменьшение (декремент) аргумента на единицу, op1:=op1-1;
- Здесь операнд op1 может быть форматов r8, m8, r16 и m16. Применение этих команд вместо соответствующих по действию команд вычитания и сложения приводит к более компактным программам. Необходимо, однако, отметить, что компактные команды **inc** op1 и **dec** op1 , в отличие от эквивалентных им более длинных команд **add** op1,1 и **sub** op1,1 никогда не меняют флаг CF.1


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Welcome to DOSBox v0.74
For a short introduction for new users type: INTRO
For supported shell commands type: HELP
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.
HAVE FUN!
The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c c:\masm4
Drive C is mounted as local directory c:\masm4\
Z:\>c:
C:\>

Процесс подготовки программы к выполнению

Программа на ассемблере MASM 4

```
data segment       ;сегмент данных программы
string db "Hello word",13,10,"$"
data ends
code segment                ;кодовый сегмент программы
      assume cs:code,ds:data
start:                      ;точка входа программы
      mov ax,data           ;загрузка в регистр DS значения
      mov ds,ax                   ;соответствующего положению
      mov dx,offset string              ;загрузка положения строки
      mov ah,09h            ;обращение за воспроизведением
      int 21h               ;строки
      mov ah,4ch            ;вызов функции завершения
      int 21h
code      ends
stack segment stack               ;стековый сегмент программы
      DW 64 DUP(?)            ;определение области стека
stack ends
      end start             ;обозначение конца и определе
                        ;ние начала
```

```
text segment
assume  Cs:text,Ds:data
begin:
        mov Ax,data
        mov Ds,Ax
        mov AH,09h
        mov Dx,offset string
        int    21h
        mov AH,4Ch
        mov  AL,0
        int   21h
text ends
datasegment
string      db "start!$"
dataends
stack       segment stack
        db  256 dup (0)
stack       ends
        end begin                    ; обозначение конца и определе
         ;ние начала
```

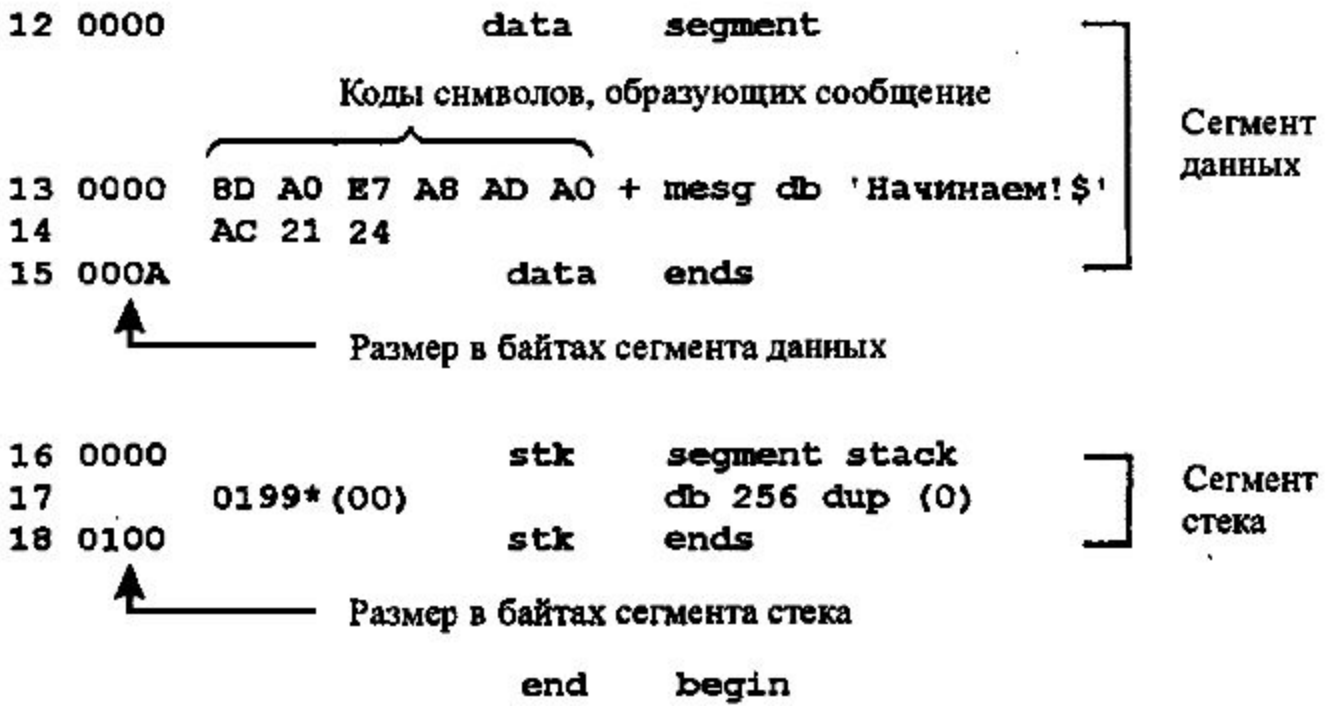
Программа на Fasm

```
org     100h            ; расположения в памяти : 100h
        start:                      ; Метка старота программы (не
          ;обязательно)
            mov ah,9                ; Функция ДОС
            mov dx,hello             ;  для вывода строки
            int 21h                 ;  на экран

            mov ah,0                ; Функция БИОС
            int 16h                 ; Ожидание нажатия клавиши
            int 20h                 ; завершение программы
        ret                         ; Возврат из процедуры start

        hello db 'Hello world!',13,10,24h
```

```
; example of simplified Win32 programming using complex macro features

include '%fasminc%\win32axp.inc'

.code

  start:
        invoke  MessageBox,HWND_DESKTOP,"Hi! I'm the example program!","Win32
        invoke  ExitProcess,0
.end start
```

- http://placeprog.com/blogs/assembler/assembler-dlja-win32-okno.html

Листинг программы


12 0000 data segment
Коды снмволов, образующих сообщение
13 0000 BD A0 E7 A8 AD A0 + mesg db 'Начинаем!$'
14 AC 21 24
15 000A data ends
Сегмент данных
Размер в байтах сегмента данных
16 0000 stk segment stack
17 0199*(00) db 256 dup (0)
18 0100 stk ends
Сегмент стека
Размер в байтах сегмента стека
end begin