Лекция №3
Тема "Представление целых чисел"

В общем случае под целое число можно отвести любое число соседних байтов памяти, но система команд поддерживает работу с числами только размером:

- байт;
- слово;
- двойное слово (частично)

Если числа занимают иное количество байтов, то все операции над ними надо реализовывать самому программисту.

Целые числа без знака

Представляются в виде байта, слова, двойного слова - в зависимости от размера.

В байте	от 0 до 255	$(2^8 - 1)$
В слове	от 0 до 65535	$(2^{16} - 1)$
В двойном слове	от 0 до 4294967297	$(2^{32} - 1)$

Числа записываются в ячейки в двоичной системе счисления и занимают все разряды ячейки.

Пример
Целое число длиной в байт содержит число 98 в десятичной системе счисления.

$98_{10} = 62_{16} = 1100010_2$

Особенность представления чисел в памяти

Числа размером в слово и двойное слово хранятся в памяти в «перевернутом виде».

Числа длиной в слово

Если на число отведено слово памяти, то старшие (левые) 8 бит числа размещаются во втором байте слова, а младшие (правые) 8 бит числа - в первом байте.

В памяти число 62h выглядит:

A	A+1
62	00

0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0

В регистрах числа размером в слово хранятся в нормальном, неперевернутом виде - за этим следят команды пересылки.

			BH	BL
62	00	BX	00	62

Данное длиной в двойное слово

В первом байте двойного слова хранятся младшие (правые) 8 битов числа, во втором байте - предпоследние 8 битов и т.д.

12345678h

A	A+1	A+2	A+3
78	56	34	12

Целые числа со знаком

Представляются в виде байта, слова, двойного слова - в зависимости от размера.

В байте	от -128 до 127	$(2^7 - 1)$
В слове	от -32768 до 32767	$(2^{15} - 1)$
В двойном слове	от -2147483648 до 2147483647	$(2^{31} - 1)$

Числа записываются в ячейки в двоичной системе счисления и занимают все разряды ячейки.

В памяти числа представлены в дополнительном коде (инверсирование битов и прибавление 1).

$$\text{Доп}(x) = \begin{cases} x, & x \geq 0 \\ 2^k - |x|, & x < 0 \end{cases}$$

Самый левый бит играет роль знакового:
0 - число положительное
1 - число отрицательное

Эти числа также хранятся в «перевернутом виде» для слова и двойного слова.

Пример

Число -98, представлено в памяти в виде слова.

1. Переводим число в двоичную систему счисления:
$98_{10} = 62_{16} = 1100010_2$

2. Записываем полученное двоичное число в слово:
00000000 01100010

3. Обратный код:
11111111 10011101

4. Дополнительный код:
11111111 10011110

5. В памяти будет представлено:

A	A+1
9E	FF

Лексемы

простейшие конструкции языка ассемблер:

- идентификаторы;
- целые числа;
- символьные данные;
- предложения;
- комментарии;
- команды;
- директивы.

Идентификаторы

используется для обозначения различных объектов программ - переменных меток, названий операций и т.п.

Идентификатор последовательность из латинских букв, цифр и следующих знаков: ? **.** @ _ $

Ограничения:
- длина может быть любой, но значащими являются только первые 31 символ;
- идентификатор не должен начинаться с цифры;
- точка может быть только первым символом (**.**A - правильно, A**.** - нет).

Целые числа

Десятичные числа (decimal)	25, -386, +4, 25d, -286d
Двоичные числа (binary)	101b, -11000b
Восьмеричные числа (octal)	74q, -46q
Шестнадцатеричные числа (hexadecimal)	1AFh, -1AFh

Примечание

Если шестнадцатеричное число начинается с буквы (A-F), то в начале числа обязательно должен стоять незначащий ноль.

`0A5h` - число, **`A5h`** - идентификатор

Символьные данные

Символьные данные заключаются в одинарные или в двойные кавычки.
Если в качестве символа или внутри строки надо указать кавычку, то:

- если символ или строка заключена в одинарные кавычки, то одинарную кавычку надо удвоить, а вот двойную кавычку не надо удваивать

```
'don''t' 'don"t'
```

- если внешние кавычки двойные, то двойная кавычка должна быть удвоена, а одинарная не удваивается

```
"don't"  "don""t"
```

Предложения

Программа на языке ассемблера - это последовательность предложений, каждое из которых записывается в отдельной строке.

Правила записи предложений:

- В строке записывается одно предложение, длина которого не может быть больше 131 символа.
- Пробел обязателен между рядом стоящими идентификаторами или числами, чтобы отделить их друг от друга.
- Там, где допустим один пробел, можно ставить любое число пробелов.

По смыслу все предложения делятся на две группы:

- комментарии;
- команды;
- директивы (приказы).

Комментарии

любая строка, начинающаяся со знака «точка с запятой»
Можно использовать многострочный комментарий. Он должен начинаться со строчки

Comment <маркер> <текст>

В качестве *маркера* берется первый за словом Comment символ, отличный от пробела. Концом такого комментария считается конец первой из последующих строк программы, в которой (в любой позиции) снова встретится этот же маркер.

```
Comment * все
это является
комментарием * и это тоже
```

Команды

символьная форма записи машинных команд.

[<метка> :] <мнемокод> [<операнды>] [;<комментарий>]

Метка служит для ссылок на команду из других мест программы.
Разрешается в одной строке указывать только метку (с двоеточием) и больше ничего. Это удобно, когда команду надо пометить двумя и более метками и когда метка очень длинная и потому часть команд сильно сдвигается вправо, что плохо смотрится.

```
Initialization:

Lab:add bx, ax
```

Мнемокод - служебное слово, указывающее в символьной форме операцию, которая должна выполнить команда.

Операнды - отделяются друг от друга запятыми. Записываются в виде выражений.

Комментарий - для пояснения именно данной команды. Начинается комментарий с точки с запятой.

Директивы - приказ ассемблеру какие константы и переменные используются программой и какие имена им дали.

Директивы определения данных

[<имя>] DN <выражение>

где DN принимает значения:

DB - определяются данные размером в байт;
DW - определяются данные размером в слово;
DD - определяются данные размером в двойное слово;
DQ - определяются данные размером в учетверенное слово;
DT - определяются данные в 10 байт.

Выражение:

1. Константа

```
Count   DB  254  ;0FEh
```

2. Знак «**?**» - неопределенное значение

```
Count   DB  ?
```

3. Несколько констант разделенных запятой. Количество ограничено только длиной строки

```
Count   DB  11, 12, 13, 14
```

В виде последовательности смежных байт. Ссылка по имени Count указывает на первую константу равную 11, Count+1 - на вторую (12) и т.д.

```
Count   DB  11
        DB  12
        DB  13
        DB  14
```

Можно повторять константы

[<имя>] DN <число повторений> DUP (выражение)

```
F1   DW   10 DUP (?)          десять неопределенных слов
F2   DB   5 DUP (14)          F2  DB14, 14, 14, 14, 14
F3   DB   3 DUP (4(DUP (8))   F3  DB8, 8, 8, 8, 8, 8, 8, 8,8, 8, 8, 8
F4   DB   2 DUP ('ab',?, 1)   F4  DB'ab', ?, 1, 'ab', ?, 1
F5   DB   6 DUP (7 DUP (?))   описание матрицы 6 x 7
```

Директива эквивалентности

Директива EQU связывает значение с идентификатором (определенным именем, вместо которого в программе ассемблер подставит указанное значение). Исключительно для числовых значений помимо директивы EQU, можно применять знак равенства (=).

```
Count       EQU  10
Element     EQU  5
Size    =   Count + Element
MyBoat      EQU  "Пример"
Size    =   0
```

Идентификаторы равенств не являются переменными - ни они, ни их значения *не содержатся* в сегменте данных программы. Команды ассемблера не могут изменить значения идентификатора, если они описаны с помощью EQU или =.