Лабораторная работа по информатике №2

1. **Цель работы**: Научиться работе с различными массивами данных. Реализовать распаковку массива и алгоритмы сортировки.

# Задание 1

В первой программе необходимо посчитать кол-во отрицательных элементов в массиве. Для массива длиной **l** алгоритм подсчета будет выглядеть следующим образом *(рис. number of negative elements.png)*:

# Листинг

* + - 1. Код программы приведен в файле **lab1.asm**.

# Анализ

Регистр ECX используется в качетсве **счетчика**, поэтому он первым делом опусташется командой XOR (стр. 6). В **EBX** загружается адрес первого эл-та (стр. 7), а в **EDI** – длина массива (стр. 8), которая вычисляется выражением **EQU $-array** в .data, где array – байтовый массив, $ - позиция начала строки, содержащей выражение. Регистр **ESI** хранит **индекс** элемента, который изначально выставляется в 0 (стр. 9).

Далее выставляется метка **Iteration** (стр. 10), начинающая новый проход. В нем значение EDI сравнивается с ESI (стр. 11). Если элементов в массиве не осталось, то алгоритм завершается (стр. 12) и выводит ECX (стр. 23). В регистр **AL** загружается текущий эл-т (стр. 14) и сравнивается с нулем (стр. 15). В случае, когда эл-т **не меньше 0**, алгоритм переходит на следующий эл-т (стр. 20-21), иначе увеличивается счетчик и загружается следующий эл-т (стр. 16-18).

# Пример

Программа для массива array = **{1, -2, 3, -4, 0}**, на выходе вывела **2**, что соответсвует реальному количеству отрицательных элементов. Таким образом программа составлена и выполнена верно.

# **Задание** 2

В данном задании программа должна распаковать массив из 8 элементов по 5 бит. Для этого сначала необходимо составить упакованный массив. Рассмотрим процесс упаковки на примере байтового массива array = **{1, 2, 3, 4, 5, 6, 7, 8}**:

1. Все элементы переводятся в **bin по 5 бит** *(00001, 00010, 00011, 00100, 00101, 00110, 00111, 01000)*
2. Полученная **цепочка двоичных чисел** соединяется и **разделяется** уже **по 8 бит** *(00001000, 10000110, 01000010, 10011000, 11101000)*
3. Для удобства все байты переводятся в hex *(0x08, 0x86, 0x42, 0x98, 0xE8)*

Для распаковки данного массива применяется метод маскирования битов. Так, зная, что каждое число предствлено 5-ю битами, на элемент с помощью логической операции **AND** накладывается маска из единиц, выделающая 5 бит нужного числа, а затем производится сдвиг регистра на n бит (n – сдвиг числа относительно первого/последнего бита).

Когда число располагается в двух байтах по частям, то применяется тот же метод, только использу уже два регистра. Первая часть заносится и сдвигается в одном регистре, вторая часть — в другом. Затем в одном из регистров биты объединяются по средством логической операции **OR**.

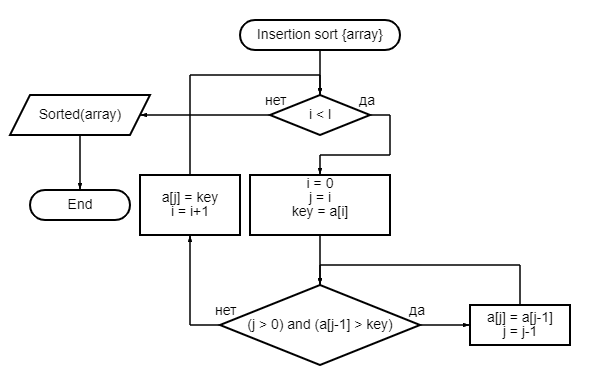
# Листинг

* + - 1. Код программы приведен в файле **lab2.asm**.

# Пример

* + - 1. В регистр **ESI** загружается адрес первого элемента (стр. 6). Затем в **AL** помещаем сам первый элемент (стр. 7). Он представлен сначала 5-ю битами первого числа, потом 3-мя битами второго. Для получения первого числа накладываем на AL маску 0xF8 (стр. 8). Так, первый элемент из 00001000 стал **00001**000.Теперь его осталось сдвинуть вправо с помощью операции **SHR** на 3 бита (стр. 9) - 000**00001**. По итогу в регистре AL хранится первое запакованное число **1**.
      2. Для распаковки второго числа загружаем в **BL** все тот же первый элемент (стр. 13). Зная, что первые 3 бита числа — это последние 3 бита элемента, накладываем маску 0x07 (стр. 14) и сдвигаем выделенные биты на 2 влево (стр. 15). Таким образом, в BL хранится начальные биты второго числа. В AL же загружаем второй элемент (стр. 16-17) и выделяем из него первые 2 бита, последние у числа, маской 0xC0 (стр. 18). Сдвигаем вправо на 6 бит (стр. 19) и объединяем (стр. 20) - OR AL, BL. При выводе AL будет значение второго числа.
      3. Аналогичным путем распаковываются и все оставшиеся числа. Данная программа будет работать для любых чисел от 0 до 31, так как метод распаковки не заисит от входных значений.

# Задание 3 (Сортировка вставками)

* + - 1. В данном задании необходимо реализовать сортировку вставками. Блок-схема алгоритма для массива **array**, и длиной **l** представлена на рисунке ниже *(рис. inserttion sort.png)*:
      2. Данная сортировка проходит входные элементы, помещает каждый рассматриваемый элемент в подходящее место среди ранее отсортированных элементов. Ее сложность O(n2).

# Листинг

* + - 1. Код программы приведен в файле **lab3.asm**.

# Анализ

В регистр ESI загружается адрес первого эл-та (стр. 7) и дублируется в EDX (стр. 8). В EDI загружается длина массива уже упомянутым способом (стр. 9). Метка **Iteration** начинает новый проход по элементам. Если ESI = EDI, то алгоритм завершается (стр. 11-12), иначе в **AL** записывается key (стр. 13), равное текущему элементу. В EBX загружается ESI (j = i) (стр. 14). Далее идет сравнение, отделяемое меткой **Comparison**. В **AH** помещается значение предыдущего элемента (стр. 16). Если AH > AL (стр. 17) и EBX > 0 (стр. 21), то предыдущий элемент поднимается на место текущего (стр. 29), и EBX декриментируется (стр. 30). Иначе текущему элементу присваевается значение key (стр. 25), и происходит переход к следующему элементу (стр. 26-27).

Для печати отсортированного массива (стр. 33-39) макросом PRINT\_DEC выводятся элементы, начиная с EDX и до EDI.

# Пример

Для массива array = **{35, 12, -8, 47, 0, 31}** программа вывела его элементы в возрастающем порядке: **{-8, 0, 12, 31, 35, 47}.** Следовательно алгоритм и программа составлены верно.

# Вывод

По выполнении работы я научился работать с массивами, реализовал распаковку массива и один из видов сортировки, а также повысил знания в работе с памятью и принципах адресации.