# 实验二 无符号数组排序

1. 题目要求

在 buf 缓冲区中存放有 50 个字节数据(无符号数)，编写程序将这些数据由小到大排序，排序后的数据仍放在该区域中。具体功能如下:

原始数据在源程序中由定义给出;

1. 在屏幕上先显示排序前的数据（十六进制），数据以每行 10 个的格式显示在屏幕上，数据与数据间用空格分开；
2. 完成排序;
3. 在屏幕上显示排序后的数据（十六进制），数据以每行 10 个的格式显示在屏幕上，数据与数据间用空格分开。

要求:

1. 如果数据个数变为 60 或 78 等的取值时，程序代码部分无需修改;
2. 编写完整程序，并能上机运行;
3. 完成实验报告(格式不限)，写出算法思想、调试过程及心得体会，2-3 页以内。
4. 算法说明

综合考虑时间复杂度、空间复杂度和编码难度，使用快速排序算法。

一轮快速排序：

* 1. 抽取首位数字x作为此轮排序基准，存在寄存器中，将[buf+0]视为空位；设置两个记录遍历到的数组位置的指针i和j，i从首位开始向后移动（初值0），j从末位开始向前移动（初值为数组长度-1）；
  2. j从后往前寻找比x小的数，找到后将符合要求的[buf+j]复制到第i位[buf+i]，则[buf+j]可视为空位，转到状态（3）；
  3. i从前往后寻找比x大的数，找到后将符合要求的[buf+i]放在第j位[buf+j]，则[buf+i]可视为空位，转到状态（2）；
  4. 重复（2）（3）直到i==j，此时[buf+i]为空位，但已经没有需要比较的数字了，且其左侧所有数都小于x，右侧所有数都大于x，将x放入空位，本轮结束。

继续在左侧、右侧两段中进行下一轮快速排序，直到数组长度=1。

1. 代码块、子程序功能

各代码块功能与流程图一一对应。

排序：

1. QSORT：快速排序函数，调用时将左边界DI至右边界BX内（左右均为闭区间）的数组从无序变为有序（从小到大）。
2. PARTITION：进行一轮排序，将左边界DI至右边界BX内（左右均为闭区间）的数据分成两段，左段均小于基准值，右段均大于基准值。其中基准值为输入时的首位。将基准值地址存在AX输出。

输出：

1. PROC SHOW：控制输出格式，每行显示十个两位16进制数字输出并以回车结尾。使用时SI指向要显示的数组。
2. PRINT：输出一个两位16进制数，并在其后添加一个空格，调用时输出内容放在DL中。

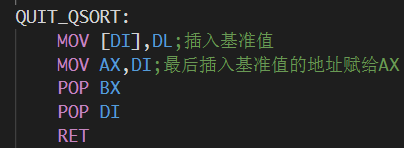
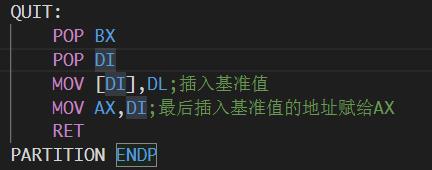
保证数据个数改变时无需修改代码：

N=$-buf

LAST=N-1

其中，$表示当前段地址，所以N可以获得buf中输入不同的数据时其总长度，而LAST可用于计算最后一位数据的位置：buf+LAST

1. 调试过程
2. 本次实验相比上次复杂许多，由于汇编语言无法模块化测试，程序没有正常执行时很难定位错误出现的位置。对此我的解决办法可总结为以下两点：
3. 时刻明确代码执行的顺序和逻辑，减少对编译器的依赖。在使用高级语言时，遇到一些模棱两可、逻辑上不太清楚的问题，例如循环的边界等；由于IDE/编辑器提供了便捷的debug环境，我常常于在编码时放弃思考，通过debug时观测运行时对象、数据的状态判断应该如何将“随意”写的代码段改对。而汇编语言的编译环境通常不方便快速迭代，因此在更需要确保逻辑清晰。
4. 学会运用中断输出关键内容和DOSBox debug中的单步执行、断点、监视（watch expression）、查看内存段等功能。虽然编译器比较原始，但也提供了这些基本的功能，学会适时使用它们。例如出现死循环时就可以使用单步执行找到循环判断出错的代码段位置；而到了程序编写比较完善时才可以通过观察输出验证逻辑是否正确。
5. 留意数据的进制类型。如果程序员将十进制和16进制弄混，程序通常仍可以看似正常地编译执行，然而结果肯定不正确。例如将INT 21H写成INT 21不会导致报错。
6. 回车≠换行。BEFORE db 'BEFORE SORT',**0dh**,'$' 输出的字符串中加入了回车字符0dh后就不显示了，原因是ASCII码中的回车(Enter/Return)指的是回到行首/归位的最传统的含义，因此输出回车后，后续内容在同一行行首重新开始输出，也就覆盖了原输出。ASCII码的换行符为0ah（换行至下一行行首）。
7. 注意子程序执行时保护现场的操作，确保恢复现场的位置在子程序全部操作执行完毕以后。例如下图左侧就是我写的bug，右图是正确的代码。



1. 心得体会
2. 初次使用汇编语言写需要多个子程序协同完成的项目，我由于不熟悉汇编语言特性写出来的函数复用性较差，也难以做到模块间低耦合，最初版本的代码各子程序间还有交叉使用寄存器数据的情况，导致修改代码时遇到很大的困难。后来我重新研究课件上的示例代码之后，加强了堆栈的使用，编码效率总算稍有提高。
3. 开始实验前，我本以为实验主要难度在于排序的过程，也借此复习了一下数据结构与算法课程中讲到的常用排序算法。然而实际编码时，困扰我最多的却是“输出”这一不起眼的过程。想不到离开了高级语言一些习以为常的函数，这些看似简单的底层实现也并不简单。我再次深刻体会到：我们日常编码都是建立在前人“造轮子”的基础上的，了解这些底层的实现对技术的发展是很有意义的。