

**微机原理与汇编语言课程设计**

学院名称： 计算机工程学院

专 业： 计算机科学与技术

班 级： 18计3z

学 号： 2018141524

姓 名： 刘展鹏

指导教师： 傅中君

2020年6月18日

目录

[一、 摘要 2](#_Toc437021299)

[二、 课程设计目的 2](#_Toc622899553)

[三、 设计任务及要求 2](#_Toc90008146)

[四、 程序总体结构及基本算法 2](#_Toc938422334)

[五、 概要设计 3](#_Toc944263970)

[1 各模块设计思想 3](#_Toc340392460)

[1.1 投骰子模块 3](#_Toc79639612)

[1.2 排序模块 3](#_Toc620646803)

[2 程序各模块流程图 3](#_Toc2108851670)

[六、 程序详细设计 5](#_Toc1399907602)

[1.数据的定义 5](#_Toc416230282)

[2.投骰子的设计 6](#_Toc1228111295)

[3.排序的设计 7](#_Toc1401203748)

[七、 测试数据及bug分析 13](#_Toc1872244083)

[八、 收获与体会 14](#_Toc1875907137)

[九、 附录(源代码) 15](#_Toc1208946388)

1. 摘要

**摘要**：用汇编语言作为自己程序设计的主流语言是不现实的，当代码量达到一定程度后，代码的维护会变困难，甚至不可能实现。但我们仍要学习汇编语言，但学习汇编的目的不是用它去进行实实在在的程序设计，而是充分获得底层的编程体验和深刻理解机器运行程序的机理。另外，对于一个人来说，不能没有常识，尽管常识不能直接挣钱吃饭，但它影响谈吐，影响你的判断力和决断力，决定着你接受新事物和新知识的程度。相应的，汇编语言就是计算机语言里面的常识和基础。

1. 课程设计目的

1.了解汇编语言程序设计的过程；熟悉DEBUG的基本调试命令；掌握应用DEBUG调试汇编语言程序的基本方法，熟悉顺序、分支、循环、子程序等设计的方法。

2.熟练掌握子程序的参数传递的方式、子程序的嵌套及其设计方法和技巧

3.培养学生创新精神和独立运用所学基本知识和技能分析问题和解决问题的能力。

4.进行汇编语言程序设计应用的综合性训练；掌握基于DOS/BIOS功能调用的标准控制台输入/输出程序的设计方法；灵活应用调试手段，实现标准控制台输入/ 输出程序的调试。

1. 设计任务及要求

本程序设计实现一个简单的投骰子，一共有三个人进行投骰子。三个人投完骰子可以进行排序。

1. 程序总体结构及基本算法

根据设计任务及要求，本程序主要分为一下模块：

* + - 1. 投骰子模块；
      2. 排序模块

1. 概要设计

1 各模块设计思想

1.1 投骰子模块

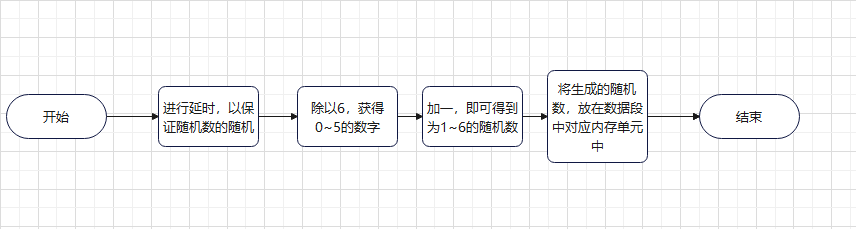
投骰子，在程序中是利用随机数来实现的，我们利用系统时钟的任意性来实现随机数的随机，利用到了1AH号时钟中断，从而获得随机数，我们利用除法来实现一个从1~6的随机数，即骰子的大小范围。

1.2 排序模块

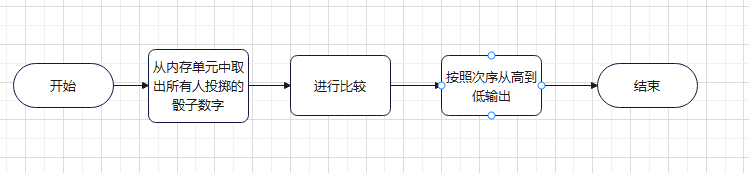
排序模块，在本程序中，就是对三个人投出的骰子进行大小比较，然后依次按照从大到小的次序进行输出，我们可以根据三个人投出的骰子的大小继续对三个人进行是否投骰子的判断，如果有两个人或者三个人的骰子的点数一样的话，可以选择重新进行投掷。

2 程序各模块流程图

生成随机数，即投骰子：



排序：



1. 程序详细设计
2. 数据的定义

data segment

sum dw 0

value1 dw 0

db '$'

value2 dw 0

db '$'

value3 dw 0

db '$'

menu0 db "---------------Throw the dice-----------------",13,10

menu1 db "---------1. The first people throw.-----------",13,10

menu2 db "---------2. The second people throw.----------",13,10

menu3 db "---------3. The third people throw.-----------",13,10

menu4 db "---------4. Sort.-----------------------------",13,10

menu5 db "---------0. Exit.-----------------------------",13,10

menu6 db "Please input number (0~4) : ",'$'

message1 db "The first people throw the dice, the number is:",'$'

message2 db "The second people throw the dice, the number is:",'$'

message3 db "The third people throw the dice, the number is:",'$'

CRLF db 13,10,'$'

data ends

数据段中定义了三个变量来存储三个人投出的骰子的数值，以及必要的输出目录的字符。

1. 投骰子的设计

; generate the random

; return value is in bx

randget proc near

call    delay

push    ax

push    dx

mov     ah,00h

int     1ah

mov     ax,dx

and     ah,3

mov     dl,6

div     dl

mov     al,ah

xor     ah,ah

inc     ax

add     ax,'0'

mov     bx,ax

pop     dx

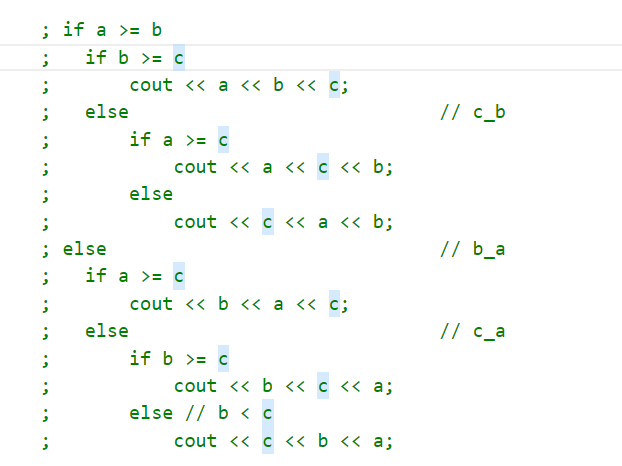
pop     ax

ret

randget endp

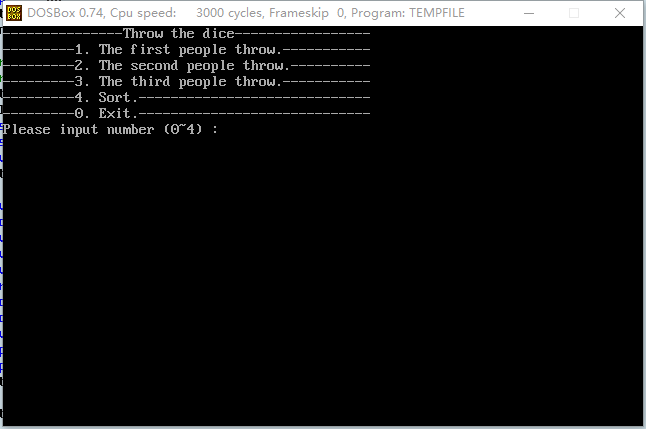
这里是利用系统时钟，1A号中断，来获得一个随机数，存放在dx里面，然后进行除法，除以6就可以获得一个0~5的随机数，然后我们加一，就可以得到一个1~6的随机数，并将其转化为ASSCII码存储在bx寄存器。

1. 排序的设计

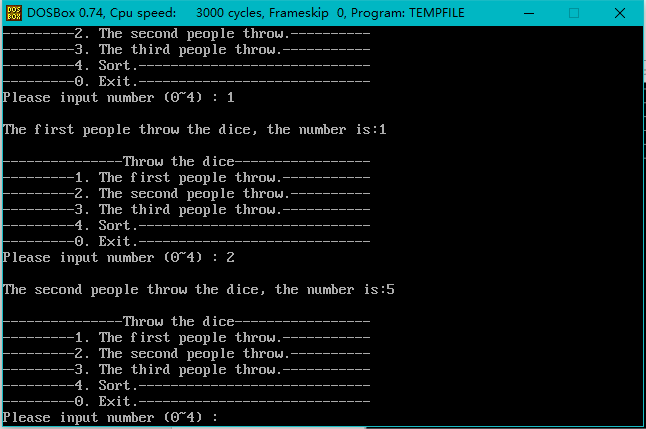


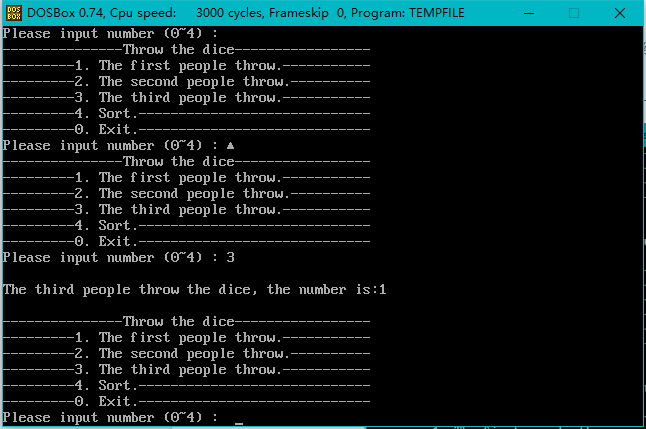
排序的大致思路就是从上面这一张图中得到，我们先将三个变量的值存放在ax、 bx、cx三个寄存器中，然后利用cmp来进行比较。

1. 测试数据及bug分析
2. 输出目录：

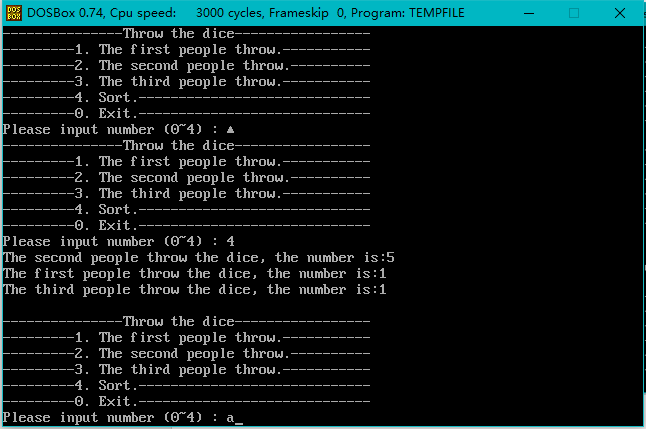


1. 投骰子：

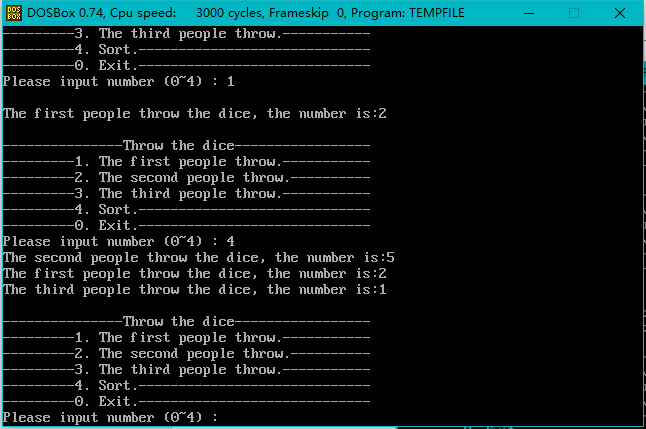




3、进行排序：



1. 让其中一个人重新进行投掷并进行排序：



1. 收获与体会

本大作业程序还有很多缺点如下：

1. 没有清屏操作
2. 查询过程是基于排序才有的
3. 代码过长，优化太少

学习计算机知识，学习的是计算机整套的系统知识,当然这其中包括的内容就多了:数据结构,操作系统，微机原理，编译原理，各种高级语言，甚至更高层次的应用,如此众多的知识对于一个初次接触计算机的学习者来说深奥,茫然.这就需要一个好的教学体系与之对应，让学生能更好的理解计算机知识.而这套教学体系的开启项就应该是汇编语言原因是由汇编语言本身的特性决定的:它充分接触硬件，理解计算机运行原理，可以让操作者感知计算机的运行过程和原理，从而能够对计算机硬件和应用程序之间的联系和交互形成一个清晰的认识.这也是最能够锻炼编程者编程思维逻辑的,只有这样，学习者才能形成一个软.硬兼备的编程知识体系，这是任何高级语言都无法给予的!相对于繁复的高级语言，汇编语言指令集合更简约，指令操作更直接,从汇编开始学习更符合循序渐进的学习原理!最根本的是让学习者知道计算机到底是什么东西，抹去计算机学习者对计算机神秘,深奥的感觉，同时增加学习者的信心。

1. 附录(源代码)

**; throw the dice three times,and sort by the number of the dice**

**assume ds:data,cs:code,ss:stacks**

**data segment**

**sum dw 0**

**value1 dw 0**

**db '$'**

**value2 dw 0**

**db '$'**

**value3 dw 0**

**db '$'**

**menu0 db "---------------Throw the dice-----------------",13,10**

**menu1 db "---------1. The first people throw.-----------",13,10**

**menu2 db "---------2. The second people throw.----------",13,10**

**menu3 db "---------3. The third people throw.-----------",13,10**

**menu4 db "---------4. Sort.-----------------------------",13,10**

**menu5 db "---------0. Exit.-----------------------------",13,10**

**menu6 db "Please input number (0~4) : ",'$'**

**message1 db "The first people throw the dice, the number is:",'$'**

**message2 db "The second people throw the dice, the number is:",'$'**

**message3 db "The third people throw the dice, the number is:",'$'**

**CRLF db 13,10,'$'**

**data ends**

**stacks segment stack**

**dw 20 dup(?)**

**stacks ends**

**; define macro : enter**

**output\_CRLF macro**

**mov dx,offset CRLF**

**mov ah,09h**

**int 21H**

**endm**

**; define macro : message**

**output\_message macro x**

**mov dx,offset x**

**mov ah,09h**

**int 21h**

**endm**

**; define macro: output the menu**

**display\_menu macro**

**mov dx,offset menu0**

**mov ah,09h**

**int 21H**

**endm**

**; define block : until input enter**

**block macro**

**block\_loop:**

**mov ah,0**

**int 16h**

**cmp ah,1ch**

**jne block\_loop**

**endm**

**code segment**

**start:**

**mov ax,data**

**mov ds,ax**

**mov ax,stacks**

**mov ss,ax**

**main:**

**display\_menu**

**mov ah,1**

**int 21H**

**push ax**

**block**

**output\_CRLF**

**pop ax**

**xor ah,ah**

**cmp ax,0030h**

**je main\_return**

**cmp ax,0031h**

**je loop\_1 ;1. The first people throw.**

**cmp ax,0032h**

**je loop\_2 ;2. The second people throw.**

**cmp ax,0033h**

**je loop\_3 ;3. The third people throw.**

**cmp ax,0034h**

**je loop\_4 ;4. Sort.**

**jmp main**

**loop\_1:**

**call randget**

**mov di,offset value1**

**mov ds:[di],bx**

**output\_CRLF**

**output\_message message1**

**mov dl,bl**

**mov ah,2**

**int 21h**

**output\_CRLF**

**output\_CRLF**

**jmp main**

**loop\_2:**

**call randget**

**mov di,offset value2**

**mov ds:[di],bx**

**output\_CRLF**

**output\_message message2**

**mov dl,bl**

**mov ah,2**

**int 21h**

**output\_CRLF**

**output\_CRLF**

**jmp main**

**loop\_3:**

**call randget**

**mov di,offset value3**

**mov ds:[di],bx**

**output\_CRLF**

**output\_message message3**

**mov dl,bl**

**mov ah,2**

**int 21h**

**output\_CRLF**

**output\_CRLF**

**jmp main**

**loop\_4:**

**call sort**

**output\_CRLF**

**jmp main**

**main\_return:**

**mov ah,4ch**

**int 21H**

**; sort the number of dices**

**sort proc near**

**push ax**

**push bx**

**push cx**

**mov ax,value1;4**

**mov bx,value2;6**

**mov cx,value3;4**

**cmp ax,bx**

**jl b\_a**

**cmp bx,cx**

**jl c\_b**

**output\_message message1**

**output\_message value1**

**output\_CRLF**

**output\_message message2**

**output\_message value2**

**output\_CRLF**

**output\_message message3**

**output\_message value3**

**output\_CRLF**

**jmp sort\_ret**

**c\_b:**

**cmp ax,cx**

**jl c\_a\_b**

**output\_message message1**

**output\_message value1**

**output\_CRLF**

**output\_message message3**

**output\_message value3**

**output\_CRLF**

**output\_message message2**

**output\_message value2**

**output\_CRLF**

**jmp sort\_ret**

**c\_a\_b:**

**output\_message message3**

**output\_message value3**

**output\_CRLF**

**output\_message message1**

**output\_message value1**

**output\_CRLF**

**output\_message message2**

**output\_message value2**

**output\_CRLF**

**jmp sort\_ret**

**b\_a:**

**cmp ax,cx**

**jl c\_a**

**output\_message message2**

**output\_message value2**

**output\_CRLF**

**output\_message message1**

**output\_message value1**

**output\_CRLF**

**output\_message message3**

**output\_message value3**

**output\_CRLF**

**jmp sort\_ret**

**c\_a:**

**cmp bx,cx**

**jl c\_b\_a**

**output\_message message2**

**output\_message value2**

**output\_CRLF**

**output\_message message3**

**output\_message value3**

**output\_CRLF**

**output\_message message1**

**output\_message value1**

**output\_CRLF**

**jmp sort\_ret**

**c\_b\_a:**

**output\_message message3**

**output\_message value3**

**output\_CRLF**

**output\_message message2**

**output\_message value2**

**output\_CRLF**

**output\_message message1**

**output\_message value1**

**output\_CRLF**

**jmp sort\_ret**

**; if a >= b**

**; if b >= c**

**; cout << a << b << c;**

**; else // c\_b**

**; if a >= c**

**; cout << a << c << b;**

**; else**

**; cout << c << a << b;**

**; else // b\_a**

**; if a >= c**

**; cout << b << a << c;**

**; else // c\_a**

**; if b >= c**

**; cout << b << c << a;**

**; else // b < c**

**; cout << c << b << a;**

**sort\_ret:**

**pop cx**

**pop bx**

**pop ax**

**ret**

**sort endp**

**; generate the random**

**; return value is in bx**

**randget proc near**

**call delay**

**push ax**

**push dx**

**mov ah,00h**

**int 1ah**

**mov ax,dx**

**and ah,3**

**mov dl,6**

**div dl**

**mov al,ah**

**xor ah,ah**

**inc ax**

**add ax,'0'**

**mov bx,ax**

**pop dx**

**pop ax**

**ret**

**randget endp**

**; delay the time**

**delay proc near**

**push dx**

**push ax**

**mov dx,0010h**

**mov ax,0**

**s1:**

**sub ax,1**

**sbb dx,0**

**cmp ax,0**

**jne s1**

**cmp dx,0**

**jne s1**

**pop ax**

**pop dx**

**ret**

**delay endp**

**code ends**

**end start**