**实验三 子程序应用程序设计**

班级： 23计科（）班 姓名： 学号：

1. **实验目的**
   1. 掌握程序设计中的子程序结构；
   2. 熟练使用过程伪指令、子程序调用和返回等汇编语言的指令编写子程序；
   3. 掌握数制转换方法；
   4. 掌握利用DOS系统功能调用进行字符输入及字符输出（显示）的方法。

**二、实验环境与设备**

计算机一台，并且安装Windows操作系统和EMU8086或macro assembler 5.0以上版本的软件。

**三、预备知识**

**1、子过程定义：利用栈传递参数时，要掌握栈帧的使用方法**

; ================================

; 子过程定义模板

; 功能： [在此填写子过程功能]

; 输入： [描述输入参数及传递方式]

; 输出： [描述返回值/输出结果]

; 破坏： [列出被修改的寄存器]

; ================================

ProcName PROC [NEAR/FAR] ; 选择NEAR(段内)或FAR(段间)调用，默认NEAR

;=== 栈帧建立与寄存器保护 ===

PUSH BP ; 1. 保存调用者的BP

MOV BP, SP ; 2. 建立当前栈帧指针

;=== 局部变量空间分配 ===

SUB SP, LocalSize ; 分配局部变量空间(LocalSize=字节数)

; 局部变量布局示例：

; [BP - 2] : WORD变量1

; [BP - 4] : WORD变量2

; [BP - 8] : DWORD变量

; 保护需要保存的寄存器（根据实际使用选择要保存的寄存器）

PUSH BX

PUSH CX

PUSH DX

PUSH SI

PUSH DI

;=== 参数访问（通过栈帧）===

; 参数布局（假设使用标准调用约定）：

; [BP + 2] : 返回地址的偏移地址（NEAR调用）

; [BP + 4] : 返回地址的段地址（FAR调用）或第一个参数

; 标准参数访问：

MOV AX, [BP + 4] ; 获取第一个参数（NEAR调用）

MOV BX, [BP + 6] ; 获取第二个参数（NEAR调用）

;=== 子过程核心逻辑 ===

; [在此编写实际功能代码]

; 使用局部变量示例：

MOV WORD PTR [BP - 2], 0 ; 初始化局部变量

MOV AX, [BP - 2] ; 读取局部变量

;=== 设置返回值 ===

; 返回值通常存储在AX

MOV AX, ResultValue ; 16位返回值

; 恢复保护的寄存器（逆序弹出）

POP DI

POP SI

POP DX

POP CX

POP BX

;=== 清理与恢复 ===

; 释放局部变量空间

MOV SP, BP ; 1. 释放局部变量（恢复SP到BP位置）

; 恢复调用者栈帧

POP BP ; 恢复调用者的BP

;=== 返回指令 ===

RET ParamSize ; 返回并清理参数空间(ParamSize=参数总字节数)

; 或使用 RET ; 如果调用者负责清理参数

ProcName ENDP

**2、堆栈传递参数调用的实例**

; 主程序

PUSH b ; 压入参数b

PUSH a ; 压入参数a

CALL Multiply ; 调用子程序

ADD SP, 4 ; 平衡堆栈（caller清理）

; 子程序（使用BP访问参数）

Multiply PROC

PUSH BP

MOV BP, SP ; 建立堆栈帧

MOV AX, [BP+4] ; 取参数a（栈顶→BP+2为返回地址）

MOV BX, [BP+6] ; 取参数b

MUL BX ; AX = a\*b

POP BP

RET ; 返回到call下一条指令

Multiply ENDP

**3、局部变量实现：通过堆栈动态分配空间，需配合BP寄存器访问**

Swap PROC

PUSH BP

MOV BP, SP

SUB SP, 4 ; 分配2个局部变量（每个16位）

; 定义局部变量位置：

; [BP-2] → 临时变量1

; [BP-4] → 临时变量2

MOV WORD PTR [BP-2], 0 ; 初始化变量

; ... 使用局部变量 ...

MOV SP, BP ; 释放局部空间（恢复SP）

POP BP

RET

Swap ENDP

1. **实验内容**

1、显示字符串。

要求：（1）定义子过程PUTS，输出包括空格在内的所有字符，直到遇到‘\0’字符（字符串末尾标记），利用寄存器BX传递要输出的字符串首地址，返回值为输出的字符个数，返回值用寄存器AX传递。

（2）定义主过程MAIN，调用PUTS过程输出两个字符串。

2、数组求和。

要求：（1）定义子过程ArraySum，计算全局变量整数（word类型）数组array的元素之和，并存入全局变量sum。

1. 定义主过程MAIN调用子过程ArraySum，测试程序功能。

3、汉诺塔问题。

要求：（1）定义递归子程序hanoi，输出盘子的移动步骤，并返回盘子的移动次数。该过程的C函数原型为：short hanoi(short n, char from, char to, char aux)。参数说明：n: 当前要移动的盘子数量，from: 起始柱子，to: 目标柱子，aux: 辅助柱子。要求用堆栈传参。

（2）定义子过程printf，实现格式化输出，能够处理字符串中包含的格式控制符（%c、%d）。

（3）定义主过程main，调用两个子过程，测试程序功能。

**五、实验要求**

1、上机前要作好充分准备，包括程序框图、源程序清单、调试步骤、测试方法、对运行结果的分析等。

2、要熟悉与实验有关的系统软件(如编辑程序、汇编程序、连接程序和调试程序等)的使用方法。在程序的调试过程中，有意识地学习及掌握debug程序的各种操作命令，以便掌握程序的调试方法及技巧。

3、程序调试完后，须由实验辅导教师在机器上检查运行结果，经教师认可后的源程序方可通过打印输出。每个实验完成后，应写出实验报告。实验报告的要求如下：

(1)实验目的：对本次实验的目的加以说明。

(2)实验内容：本次实验你所完成的具体内容加以说明。

(3)设计思想：绘制程序框图，并说明原理及算法、程序及数据结构、主要符号名等。

(4)程序代码：经辅导教师确认后的程序清单。

(5)结果分析：包括调试情况，如上机时遇到的问题及解决办法、观察到的现象及其分析．对程序设计技巧的总结及分析；程序的输出结果及对结果的分析；实验的心得体会等。

实验报告

班级： 23计科（）班 姓名： 盲灯 学号：

1. **实验目的**
   1. 掌握程序设计中的子程序结构；
   2. 熟练使用过程伪指令、子程序调用和返回等汇编语言的指令编写子程序；
   3. 掌握数制转换方法；
   4. 掌握利用DOS系统功能调用进行字符输入及字符输出（显示）的方法。

**二、实验内容**

1、显示字符串。

程序从 main proc 入口开始执行，分为以下几个阶段：

（1）数据段与堆栈初始化

程序从 main proc 开始执行，首先将 data 段地址加载至 DS，再初始化 SS:SP 指向 stack 段顶部，为后续过程调用提供栈空间。

mov ax, data

mov ds, ax

mov ax, stack

mov ss, ax

mov sp, 128

（2）调用 PUTS 子过程输出字符串

使用 mov bx, offset str1 和 mov bx, offset str2 将两个字符串地址分别送入 BX 作为参数，调用子过程 PUTS 输出。

每次输出后，为了格式美观，使用 int 21h 的 AH=2 功能号分别输出回车13和换行10字符，确保两个字符串各占一行。

mov bx, offset str1

call puts

mov ah, 2

mov dl, 13

int 21h

mov dl, 10

int 21h

（3）PUTS 子过程的实现

puts 子过程采用标准帧结构，利用寄存器 BX 传入字符串地址，通过循环依次读取并输出字符，直到遇到终止符 0，即 '\0'。使用中断 int 21h / ah = 2 显示字符，同时使用 word ptr [bp-2] 实现字符计数，最终返回输出字符个数于 AX 中。

cmp byte ptr [bx], 0

je loop\_exit

mov dl, [bx]

mov ah, 2

int 21h

inc bx

inc word ptr [bp-2]

（4）提示信息与程序终止

程序输出两个字符串后，再显示提示信息 prompt以 $ 结尾，使用 int 21h / ah = 9 实现 DOS 风格字符串输出。随后调用 int 21h / ah = 1 等待用户按键，最后以 int 21h / ah = 4Ch 结束程序。

mov ah, 9

mov dx, offset prompt

int 21h

mov ah, 1

int 21h

mov ax, 4C00h

int 21h

2、数组求和。

要求：（1）定义子过程ArraySum，计算全局变量整数（word类型）数组array的元素之和，并存入全局变量sum。

1. 定义主过程MAIN调用子过程ArraySum，测试程序功能。

程序从 main proc 入口开始执行，分为以下几个阶段：

（1）数据段与堆栈初始化

程序起始于 main proc。首先初始化段寄存器：

mov ax, data

mov ds, ax

mov ax, stack

mov ss, ax

mov sp, 128

此操作确保数据段变量如 array、length、sum 以及提示字符串能被正常访问，同时设置栈顶地址，准备过程调用。

（2）调用数组求和子过程 arraysum

程序中使用：

push offset array ; 数组地址入栈，参数1

push sum ; 累加结果地址入栈，参数2

call arraysum ; 调用子过程

add sp,4 ; 参数出栈清理

这段代码将数组首地址和存放结果的变量地址传入 arraysum。在子过程内部通过 [bp+6] 和 [bp+4] 访问参数。

（3）实现数组求和逻辑（子过程 arraysum）

在子过程 arraysum 内部：

设置帧指针并保存现场；

使用 cx 控制循环次数（数组长度）；

使用 di 作为指针访问数组；

使用 dx 累加结果；

遍历数组并累加每个 word 型元素；

将最终和保存在 ax 中作为返回值。

核心逻辑如下：

mov di, [bp+6] ; array 起始地址

mov si, [bp+4] ; sum 变量地址

mov dx, si ; 初始化 dx = 0，后续累加

随后以 loop\_start 标签控制循环结构。

（4）输出结果字符串与结果数值

输出字符串 "Sum: $" 使用标准的 DOS 输出服务（AH=9）：

mov ah,9

mov dx,offset output

int 21h

数值输出部分调用了 print\_unsigned 子过程，该过程通过除10取余法将整数转为字符，并反序输出：

call print\_unsigned

（5）结束程序前提示与退出

程序最后输出 "press any key..." 提示信息并等待用户键入：

mov ah,9

mov dx,offset key

int 21h

mov ah,1

int 21h

最终调用 int 21h 的 4Ch 功能号返回 DOS：

mov ax,4C00h

int 21h

3、汉诺塔问题。

要求：（1）定义递归子程序hanoi，输出盘子的移动步骤，并返回盘子的移动次数。该过程的C函数原型为：short hanoi(short n, char from, char to, char aux)。参数说明：n: 当前要移动的盘子数量，from: 起始柱子，to: 目标柱子，aux: 辅助柱子。要求用堆栈传参。

（2）定义子过程printf，实现格式化输出，能够处理字符串中包含的格式控制符（%c、%d）。

（3）定义主过程main，调用两个子过程，测试程序功能。

程序从 main proc 入口开始执行，分为以下几个阶段：

（1）数据段与堆栈初始化  
程序从 main proc 开始执行，首先将 data 段地址加载至 DS，再将 stack 段地址加载至 SS，并将 SP 初始化为 128，为后续过程调用和局部变量存储提供足够的栈空间。该初始化步骤为标准段寄存器设置流程，确保程序中 mov、push、call 等操作能够正确访问数据段与栈帧空间。

mov ax, data

mov ds, ax

mov ax, stack

mov ss, ax

mov sp, 128

（2）读取用户输入，判断合法性  
程序提示用户输入盘子数，采用 int 21h / AH=1 逐字符读取方式，支持多位十进制输入，直到检测到回车AL=13结束。过程中对每一位进行合法性判断，即是否为 '0'~'9'，若非法则忽略。读取完成后通过乘 10 累加实现整数构建，最终存入 BX 寄存器。若用户输入小于 1，则提示错误信息并调用 int 21h / AH=4Ch 退出程序。

mov ah, 1

int 21h

cmp al, 13

je input\_done

（3）调用 printf2 格式化输出汉诺塔标题  
程序使用自定义 printf2 子过程模拟 C 语言的格式化打印，支持 %d 和 %c 两种基本格式。首先压栈字符串地址与整数参数 n，再调用 printf2 输出 “Hanoi Tower Solution (%d disks):”。为美观，输出后接一行分割线 ==========================。

push bx

push offset main\_str1

call printf2

（4）调用递归子过程 hanoi 实现汉诺塔逻辑  
使用 push 将参数依次压栈：盘数 n、起始柱 'A'、目标柱 'C'、辅助柱 'B'。然后调用 hanoi 子过程。该过程采用典型递归分解策略：当 n == 1 时直接输出一次移动；否则递归调用三次分别将 n-1 盘子从起点柱移至辅助柱，移动最大盘子，再将 n-1 盘子从辅助柱移至终点柱。每次调用结束后返回移动总步数并累加，最后保存在 AX 中返回给主程序。

push word ptr [bp+8] ; to

push word ptr [bp+6] ; from

push word ptr [bp+4] ; n

mov ax, offset constant\_string1

push ax

call printf2

（5）输出移动总数与退出提示  
主程序接收递归返回的总步数 BX，输出 “Total moves: ”，然后调用 print\_unsigned 子过程实现十进制整数输出。最后程序显示提示信息 press any key...，等待用户按任意键结束，调用 int 21h / AH=4Ch 正常返回。

mov ah, 9

mov dx, offset main\_str3

int 21h

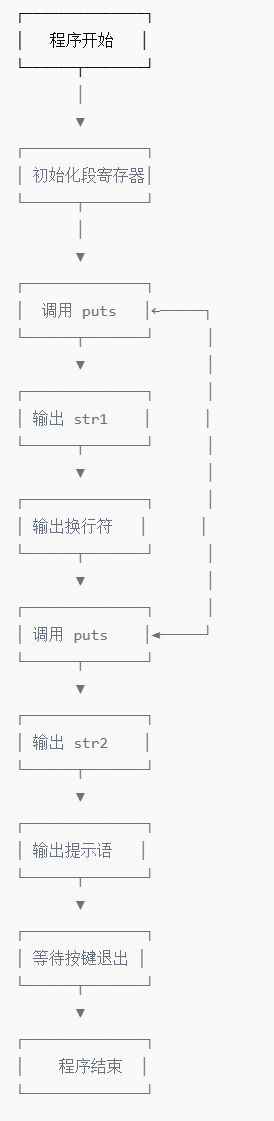
call print\_unsigned

mov ax, 4C00h

int 21h

1. 设计思想：
2. 显示字符串。

（1）程序流程图



（2）C语言伪代码

#include <stdio.h>

unsigned int puts(const char\* str) {

    unsigned int count = 0;

    while (\*str != '\0') {

        putchar(\*str);

        str++;

        count++;

    }

    return count;

}

int main() {

    char str1[] = "hello world!";

    char str2[] = "good morning!";

    puts(str1);

    putchar('\n');

    puts(str2);

    putchar('\n');

    printf("press any key to continue...");

    getchar();

    return 0;

}

（3）程序算法结构

程序分为两个核心模块：

①.主程序结构

初始化段寄存器 DS 和 SS；

使用 offset 加载字符串地址到 BX；

分别调用 puts 显示 str1 和 str2；

通过 int 21h 中断显示提示信息 prompt；

等待按键，正常退出程序。

②.子过程 puts

使用 BX 传入字符串地址；

循环读取 [BX] 内容，判断是否为 0；

通过 int 21h / ah=2 显示字符；

每次 BX++，并统计输出字符数，最终返回 AX。

（4）数据结构与寄存器说明

| 名称 | 类型 | 功能说明 |
| --- | --- | --- |
| BX | 16位寄存器 | 存放字符串起始地址 |
| [BX] | 字节 | 当前待输出的字符（字符串按字节遍历） |
| AX | 16位寄存器 | 用作 puts 的返回值，表示输出字符总数 |
| DX | 8位寄存器 | 用于 DOS 中断 int 21h / ah=2 的字符传递寄存器 |
| prompt | 数据段常量 | 字符串提示信息，使用 $ 结尾以适配功能09h |
| stack | 栈段变量 | 保存 BP 和 DX 以及局部变量 count |

（5）关键符号说明

| 符号名 | 类型/说明 | 作用 |
| --- | --- | --- |
| main | 主过程入口 | 程序起始，控制整体流程 |
| puts | 子过程 | 功能等价于 C 的 puts()，接收 BX 指针参数 |
| prompt | 字符串常量 | 用于 int 21h/09h 输出提示 |
| str1 | 字符串常量 | 第一个待输出字符串 "hello world!" |
| str2 | 字符串常量 | 第二个待输出字符串 "good morning!" |
| BP | 栈帧基地址寄存器 | 保存和访问局部变量 |

1. 数组求和。

（1）程序流程图



（2）C语言伪代码

#include <stdio.h>

unsigned int arraysum(const short\* array, int length) {

    unsigned int sum = 0;

    for (int i = 0; i < length; i++) {

        sum += array[i];

    }

    return sum;

}

void print\_unsigned(unsigned int value) {

    if (value == 0) putchar('0');

    else {

        char buffer[6];

        int i = 0;

        while (value > 0) {

            buffer[i++] = (value % 10) + '0';

            value /= 10;

        }

        while (--i >= 0) putchar(buffer[i]);

    }

}

int main() {

    short array[] = {10, 20, 30, 40, 50};

    unsigned int sum = arraysum(array, 5);

    printf("Sum: ");

    print\_unsigned(sum);

    printf("\r\npress any key...");

    getchar();

    return 0;

}

（3）程序算法结构

① 主程序结构

初始化 ds、ss、sp 等段寄存器；

将数组地址和 sum 变量地址通过 push 压入栈；

调用 arraysum 子过程；

调用 print\_unsigned 输出结果；

使用 int 21h 显示提示字符串 key；

等待按键，调用 int 21h/ah=4Ch 正常退出程序。

② arraysum 子过程

使用 [bp+6] 取得数组起始地址；

使用 [bp+4] 取得结果变量地址；

使用 di 指向数组，cx 控制循环次数；

每次 mov ax, [di] 后 add dx, ax；

最后返回总和于 ax 中。

③ print\_unsigned 子过程

使用除10法，将数值各位依次压栈；

栈顶出栈逐个字符输出，加上 '0' 变成 ASCII 数字；

使用 int 21h/ah=2 输出单字符。

（4）数据结构与寄存器说明

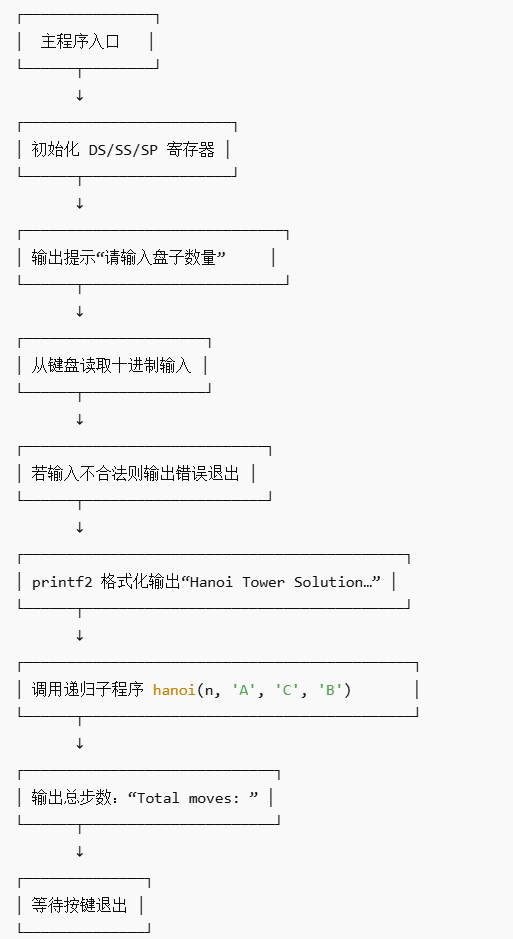
| 名称 | 类型 | 功能说明 |
| --- | --- | --- |
| BX | 16位寄存器 | 传递数组地址或字符串地址 |
| DI | 16位寄存器 | 数组指针（遍历 array） |
| SI | 16位寄存器 | sum 变量地址 |
| AX | 16位寄存器 | 运算临时寄存器；保存子过程返回值 |
| DX | 16位寄存器 | 累加寄存器；或用作字符输出时传给 DL（8位） |
| CX | 16位寄存器 | 控制数组长度的循环次数 |
| [bp+6] | 栈地址 | 参数1：array 地址 |
| [bp+4] | 栈地址 | 参数2：sum 存储地址（可选） |
| output | 数据段常量 | 输出字符串 "Sum: $" |
| key | 数据段常量 | 提示信息，含换行及 $ 结尾用于 int 21h/9 |

（5）关键符号说明

| 符号名 | 类型/说明 | 作用说明 |
| --- | --- | --- |
| main | 主过程入口 | 程序执行起点 |
| arraysum | 子过程 | 接收数组地址和长度，返回累加值 |
| print\_unsigned | 子过程 | 将整数转换为十进制字符输出 |
| array | 数据段变量 | 输入数组（5 个 word 元素） |
| sum | 数据段变量 | 存储数组求和结果 |
| length | 数据段常量 | 数组长度 = 5 |
| output | 数据段常量 | 输出字符串："Sum: $" |
| key | 数据段常量 | 提示字符串："press any key..." |
| BP | 栈帧寄存器 | 子过程内部访问参数和局部变量 |

1. 汉诺塔问题。

（1）程序流程图



（2）C语言伪代码

#include <stdio.h>

*// 汉诺塔递归函数*

short hanoi(short n, char from, char to, char aux) {

*// 基准情况：只有一个盘子时直接移动*

    if (n == 1) {

        printf("Move disk %d from %c to %c\n", n, from, to);

        return 1;

    }

 short count;

*// 递归步骤1：将n-1个盘子从起始柱移动到辅助柱*

    count=hanoi(n - 1, from, aux, to);

*// 移动第n个盘子（最大的盘子）*

    printf("Move disk %d from %c to %c\n", n, from, to);

    count++;

*// 递归步骤2：将n-1个盘子从辅助柱移动到目标柱*

    count+=hanoi(n - 1, aux, to, from);

    return count;

}

int main() {

    int n;

    printf("Please enter the number of disks for the Hanoi Tower:");

    scanf("%d", &n);

*// 验证输入有效性*

    if (n < 1) {

        printf("Error: The number of disks must be greater than 0.\n");

        return 1;

    }

    printf("\nHanoi Tower Solution (%d disks):\n", n);

    printf("==========================\n");

*// 调用递归函数*

*// A: 起始柱, C: 目标柱, B: 辅助柱*

    short count=hanoi(n, 'A', 'C', 'B');

    printf("==========================\n");

 printf("Total moves: %d\n",count );

    return 0;

}

（3）程序算法结构

程序分为两个核心模块：

① 主程序结构

初始化段寄存器 DS 与 SS，设置 SP；

输出字符串提示信息，提示用户输入盘子数量；

使用 int 21h / AH=1 实现键盘字符读取，转换为十进制数值累加到 BX；

验证输入有效性，若非法则使用 int 21h / AH=9 输出错误并退出；

采用 printf2 子过程输出格式化字符串；

调用 hanoi 递归函数，执行搬移逻辑；

最后显示移动总次数，并通过 print\_unsigned 以十进制打印结果；

显示提示并等待用户按键退出。

② 子过程 hanoi

使用标准栈帧结构 push bp / mov bp, sp；

当 n == 1 时，直接调用 printf2 输出单次移动；

否则递归三步操作：

hanoi(n-1, from, aux, to)

输出移动当前最大盘子

hanoi(n-1, aux, to, from)

使用 [bp-2] 存储递归累计步数；

返回总步数作为 AX 的返回值。

（4）数据结构与寄存器说明

| 名称 | 类型/位宽 | 功能说明 |
| --- | --- | --- |
| BX | 16位寄存器 | 存储输入盘子数量、作为递归步数累加器 |
| [BX] | 字节 | 用于读取键盘字符（转换为数字） |
| AX | 16位寄存器 | 用于输入中间计算结果、函数调用的返回值 |
| DX | 8位寄存器 | int 21h / AH=2 时的字符输出寄存器 |
| BP | 栈帧基地址寄存器 | 访问子程序参数及局部变量 |
| SP | 栈指针寄存器 | 管理栈帧的入栈与出栈 |
| constant\_string1 | 数据段字符串 | "Move disk %d from %c to %c" 用于移动操作的格式化输出 |
| main\_str1、main\_str2、main\_str3 | 数据段字符串 | 多处格式化信息字符串，用于格式化输出 |

（5）关键符号说明

| 符号名 | 类型/说明 | 作用 |
| --- | --- | --- |
| main | 主过程入口 | 控制整体流程，包括提示输入、调用递归函数 |
| hanoi | 子过程 | 实现汉诺塔递归搬运逻辑，返回移动总步数 |
| printf2 | 子过程 | 模拟 printf 实现 %d %c 输出格式化 |
| print\_unsigned | 子过程 | 打印无符号十进制数值 |
| constant\_string1 | 字符串常量 | 格式化输出每一步移动的格式模板 |
| main\_str1 | 字符串常量 | 标题字符串，含 %d 占位符用于盘数 |
| main\_str2 | 字符串常量 | 分隔线字符串 |
| main\_str3 | 字符串常量 | “Total moves:” 输出信息 |
| pkey | 字符串常量 | 提示按任意键结束 |

1. 程序代码：
2. 显示字符串。

data segment       ;全局变量

  prompt db "press any key to continue...$"

  str1 db "hello world!",13,10,0

  str2 db "good morning!",13,10,0

data ends

stack segment

    dw 128 dup(0)

 ends

;#include <stdio.h>

code segment

assume cs:code,ds:data ，ss:stack

;int main() {

main proc far

    mov ax,data

    mov ds,ax

    mov ax,stack

    mov ss,ax

    mov sp,128

    ;char str1[] = "Hello, world!";

    ;char str2[] = "Assembly is powerful.";

    ;输出两个字符串

    mov bx,offset str1   ;offset操作符：取变量名对应的地址

    call puts

    ;Puts(str1);   *// 输出第一个字符串*

    ;putchar('\n');

      mov ah,2

     mov dl,13

     int 21h

     mov dl,10

     int 21h

     mov bx, offset str2

     call puts

    ;Puts(str2);   *// 输出第二个字符串*

    ;putchar('\n');

     mov ah,2

     mov dl,13

     int 21h

     mov dl,10

     int 21h

     mov ah,9

     mov dx,offset prompt

     int 21h

      ;return 0;

     mov ah,1

     int 21h

     ;mov ah,4ch

     ;mov al,0

     ;等价于

     mov ax,4c00h

     int 21h

;}

main endp

      ;*// Puts 函数：输出以 '\0' 结尾的字符串*

;*// 参数：const char\* str（通过 BX 传地址）*

;*// 返回值：输出字符个数（返回在 AX 中）*

;unsigned int Puts(char\* str) {

puts proc near

     ;建立栈帧

    push bp ;保存bp

    mov bp,sp

    ;unsigned int count = 0;

    sub bp,2 ;栈中留出两个字节存放局部变量 count 的值,[bp-2[访问内存中的局部变量count

    push dx

    mov word ptr [bp-2],0   ;ptr操作符：显示重载操作数地址类型

loop\_entrance:

    cmp byte ptr [bx],0

    ;while (\*str != '\0') {

    je  loop\_exit

    mov ah,2

    mov dl,[bx]

    int 21h

    ;putchar(\*str);

    inc bx

    ;str++;

     ;count++;

    inc word ptr [bp-2]

    jmp loop\_entrance

    ;}

loop\_exit:

    ;return count;

    mov ax,[bp-2]

    pop dx

    add bp,2

    mov sp,bp

    pop bp

    ret

;}

puts endp

code ends

end main

1. 数组求和。

;*#include <stdio.h>*

data segment

;// 定义全局变量

;short array[] = {10, 20, 30, 40, 50};  // word类型数组

array dw 10,20,30,40,50

length dw 5

output db "Sum: $"

;short sum;                            // 存储求和结果

sum dw 0

key db 13,10,"press any key...$"

data ends

stack segment

stack ends

dw   128  dup(0)

code segment

assume cs:code,ds:data

 ;  // 主过程

;int main() {

main proc far

     mov ax,data

    mov ds,ax

    mov ax,stack

    mov ss,ax

    mov sp,128

    ;// 调用子过程

    ;ArraySum();

    push offset array

    push sum

    call arraysum

    add sp,4

    mov bx,ax

    ;// 测试输出结果

    ;printf("Sum: %d\n", sum);

    mov ah,9

    mov dx,offset output

    int 21h

    mov ax,bx

    call print\_unsigned

    add sp,2

    mov ah,9

    mov dx,offset key

    int 21h

    mov ah,1

    int 21h

    ;return 0;

    mov ax,4c00h

    int 21h

;}

main endp

print\_unsigned proc near

    push bp

    mov bp, sp

    push ax

    push bx

    push dx

    push cx

    xor cx, cx          ; 位数统计器

    mov bx, 10          ; 除数 = 10

.divide\_loop:

    xor dx, dx

    div bx              ; AX ÷ 10 → 商AX，余数DX

    push dx             ; 把余数保存，先入栈（反序）

    inc cx              ; 统计位数

    cmp ax, 0

    jne .divide\_loop

.output\_loop:

    pop dx

    add dl, '0'         ; 数字转字符

    mov ah, 2

    int 21h

    loop .output\_loop

    pop cx

    pop dx

    pop bx

    pop ax

    pop bp

    ret

print\_unsigned endp

;// 子过程：计算数组元素之和

;void ArraySum() {

arraysum proc near

    ;// 使用指针遍历数组

      ;push offset array     [bp+6]

    ;push sum                [bp+4]

    push bp

    mov bp,sp

    push bx

    push dx

    push cx

    ;short \*ptr = array;

    ;sum = 0;

    ;// 循环次数由数组长度决定

    ;for(int i = 0; i < sizeof(array)/sizeof(array[0]); i++) {

    mov bx,offset length

    mov cx,[bx]

    mov di,[bp+6]

    ;mov ax,[di]

    mov si,[bp+4]

    mov dx,si

loop\_start:

     mov ax,[di]

     add dx,ax

        ;sum += \*ptr;  // 累加当前元素

        ;ptr++;        // 移动指针到下一个元素

     add di,2

    loop loop\_start

    mov ax,dx

  ;  }

  pop cx

    pop dx

    pop bx

 mov sp,bp

 pop bp

 ret

;}

arraysum endp

code ends

end main

1. 汉诺塔问题。

; multi-segment executable file template.

data segment

    tip\_input db "Please enter the number of disks for the Hanoi Tower:$"

    error\_data db "Error: The number of disks must be greater than 0.",13,10,0

    main\_str1 db 13,10,"Hanoi Tower Solution (%d disks):",13,10,0

    main\_str2 db "==========================",13,10,"$"

    pkey db 13,10,"press any key...$"

    constant\_string1 db "Move disk %d from %c to %c",13,10,0

    main\_str3 db "Total moves:  $"

data ends

stack segment

    dw   128  dup(0)

stack ends

code segment

    assume cs:code,ds:data,ss:stack

main proc far

; set segment registers:

    mov ax, data

    mov ds, ax

    mov ax,code

    mov ax,stack

    mov ss,ax

    mov sp,128

;*#include <stdio.h>*

;int main() {

    mov ah,9

    mov dx,offset tip\_input

    int 21h

    ;printf("Please enter the number of disks for the Hanoi Tower:");

    mov bx,0

    ;int n;

    ;scanf("%d", &n);

xor bx, bx

input\_loop:

    mov ah, 1         ; AH=1: 读取一个字符（阻塞式）

    int 21h

    cmp al, 13        ; 判断是否为回车（Enter 键）

    je input\_done     ; 若是，输入完成

    cmp al, '0'

    jb  input\_loop    ; 小于 '0'，非法，忽略

    cmp al, '9'

    ja  input\_loop    ; 大于 '9'，非法，忽略

    sub al, '0'       ; 字符转数值（0~9）

    xor ah, ah        ; 清高位，AL → AX

    push ax

    mov ax,bx

    ; BX = BX \* 10 + AX

    mov cx, 10

    mul cx

    pop bx            ; AX = AX \* 10

    add ax,bx

    mov bx,ax

    jmp input\_loop

input\_done:

    cmp bx,1

    jge step\_entrance

    ;// 验证输入有效性

    ;if (n < 1) {

     mov ah,9

     mov dx,offset error\_data

     int 21h

     mov ax,4c01h

     int 21h

        ;printf("Error: The number of disks must be greater than 0.\n");

       ;return 1;

   ;}

step\_entrance:

    ;printf("\nHanoi Tower Solution (%d disks):\n", n);

    push bx

    push offset main\_str1

    call printf2

    add sp,4

    ;printf("==========================\n");

    mov ah,9

    mov dx, offset main\_str2

    int 21h

    ;// 调用递归函数

    ;// A: 起始柱, C: 目标柱, B: 辅助柱

    ;short count=hanoi(n, 'A', 'C', 'B');

     mov ax,'B'

     push ax

     mov ax,'C'

     push ax

     mov ax,'A'

     push ax

     push bx

     call hanoi

     add sp,8

     mov bx,ax

    ;printf("==========================\n");

     mov ah,9

    mov dx, offset main\_str2

    int 21h

 ;printf("Total moves: %d\n",count );

 mov ah,9

    mov dx, offset main\_str3

    int 21h

     mov ax, bx        ; 把 BX 的数值传给 AX，AX 是 print\_unsigned 的输入

     call print\_unsigned

    ;return 0;

;}

exit\_program:

     mov ah, 9

    mov dx, offset pkey

    int 21h

    mov ah, 1

    int 21h

    mov ax, 4C00h

    int 21h

main endp

;// 汉诺塔递归函数

;short hanoi(short n, char from, char to, char aux) {

hanoi proc near

    ;// 基准情况：只有一个盘子时直接移动

    ;建议栈帧

    push bp

    mov bp,sp

    sub sp,2 ;栈中留出局部变量空间

    ;寄存器保存

    push ax

    push bx

    push cx

    push dx

    cmp word ptr [bp+4],1

    jne disk\_number\_greater\_than\_1

        ;if (n == 1) {

         push word ptr [bp+8]

         push word ptr [bp+6]

         push word ptr [bp+4]

         mov ax,offset constant\_string1

         push ax

         call printf2

         ;清栈，释放参数空间

         add sp,8

        ;printf("Move disk %d from %c to %c\n", n, from, to);

        ;return 1;返回main,返回值=1

         mov ax,1

         pop dx

         pop cx

          pop bx

         pop ax

         mov sp,bp

         pop bp

         ret

   ; }

disk\_number\_greater\_than\_1:

 ;short count;

   ; // 递归步骤1：将n-1个盘子从起始柱移动到辅助柱

    ;count=hanoi(n - 1, from, aux, to);

     push word ptr [bp+8]

     push word ptr [bp+10]

     push word ptr [bp+6]

     mov ax,[bp+4]

     dec ax

     push ax

     call hanoi

     add sp,8

     mov word ptr [bp-2],ax

    ;// 移动第n个盘子（最大的盘子）

    ;printf("Move disk %d from %c to %c\n", n, from, to);

    push word ptr [bp+8]

         push word ptr [bp+6]

         push word ptr [bp+4]

         mov ax,offset constant\_string1

         push ax

         call printf2

         ;清栈，释放参数空间

         add sp,8

     inc word ptr [bp-2]

    ;count++;

    ;// 递归步骤2：将n-1个盘子从辅助柱移动到目标柱

    push word ptr [bp+6]

    push word ptr [bp+8]

    push word ptr [bp+10]

    mov ax,[bp+4]

    dec ax

    push ax

    call hanoi

    add sp,8

    add word ptr [bp-2],ax

    ;count+=hanoi(n - 1, aux, to, from);

    ;return count;

        mov ax, word ptr [bp-2]

     pop dx

    pop cx

    pop bx                   ; 把 SI 的值（总步数）放入 AX 作为返回值

    ;pop ax

    mov sp, bp

    pop bp

    ret

;}

hanoi endp

printf2 proc near

    push bp

    mov bp, sp

    sub sp, 2               ; <<< 为局部变量"参数指针"开辟空间 [bp-2]

    push di

    push bx                 ; (DI在这里实际没用了，但保留是好习惯)

    mov si, [bp+4]          ; SI <- 格式字符串地址, 这部分不变

    mov word ptr [bp-2], 6  ; <<< 初始化"参数指针", 指向第一个额外参数(n)的偏移量

next\_char:

    mov al, [si]

    cmp al, 0

    je done\_printf

    cmp al, '%'

    jne print\_normal\_char

    ; 是 '%' -> 检查格式类型

    inc si

    mov al, [si]

    cmp al, 'd'

    je output\_integer

    cmp al, 'c'

    je output\_char

    jmp continue\_loop      ; 如果是 %% 或其他不支持的，直接跳过

output\_integer:

    mov bx, [bp-2]          ; 1. 将"参数指针"的当前偏移量(比如6)加载到 bx

    mov di,bx         ; 2.  从栈上正确地取出参数值

    mov ax,[bp+di]

    push ax                 ; (暂存ax,因为print\_unsigned会修改它)

    call print\_unsigned

    pop ax

    add word ptr [bp-2], 2  ; 3. 将"参数指针"向前移动2字节,指向下一个参数

    jmp continue\_loop

output\_char:

    mov bx, [bp-2]          ; 1. 获取当前参数的偏移量

    mov di,bx

    mov dl, byte ptr [bp+di] ; 2. 从栈上正确地取出参数(字节)

    mov ah, 2

    int 21h

    add word ptr [bp-2], 2  ; 3. "参数指针"前移

    jmp continue\_loop

print\_normal\_char:

    mov dl, al

    mov ah, 2

    int 21h

    jmp continue\_loop

continue\_loop:

    inc si

    jmp next\_char

done\_printf:

    pop bx

    pop di

    pop si

    mov sp, bp      ; 恢复sp, 回收局部变量空间

    pop bp

    ret

printf2 endp

print\_unsigned proc near

    push bp

    mov bp, sp

    push ax

    push bx

    push dx

    push cx

    xor cx, cx          ; 位数统计器

    mov bx, 10          ; 除数 = 10

.divide\_loop:

    xor dx, dx

    div bx              ; AX ÷ 10 → 商AX，余数DX

    push dx             ; 把余数保存，先入栈（反序）

    inc cx              ; 统计位数

    cmp ax, 0

    jne .divide\_loop

.output\_loop:

    pop dx

    add dl, '0'         ; 数字转字符

    mov ah, 2

    int 21h

    loop .output\_loop

    pop cx

    pop dx

    pop bx

    pop ax

    pop bp

    ret

print\_unsigned endp

code ends

end main ; set entry point and stop the assembler.

1. 结果分析：
2. 显示字符串。

（1）调试情况与问题解决

在实际调试过程中，程序总体结构规范、功能分明，实现目标清晰，但也遇到了若干典型问题，具体如下：

① 数据段初始化缺失导致字符串输出错误

问题表现： 若未在 main 程序入口处执行 mov ax, data 和 mov ds, ax 操作，puts 子程序所访问的字符串地址将是无效的，导致输出乱码或程序直接异常中止。

解决方法： 严格按照初始化流程，程序开头完成段寄存器的正确赋值，确保数据段 DS 映射到 data 区域。

② 栈帧使用不当导致返回值或寄存器污染

问题表现： 若 puts 子过程没有妥善保存调用现场（如未压栈 DX，未恢复 BP），可能在调用返回后引发主程序行为异常或参数错误。

解决方法： 子过程中规范建立栈帧 push bp / mov bp, sp，调用结束前完整恢复 sp 与寄存器现场，避免栈紊乱。

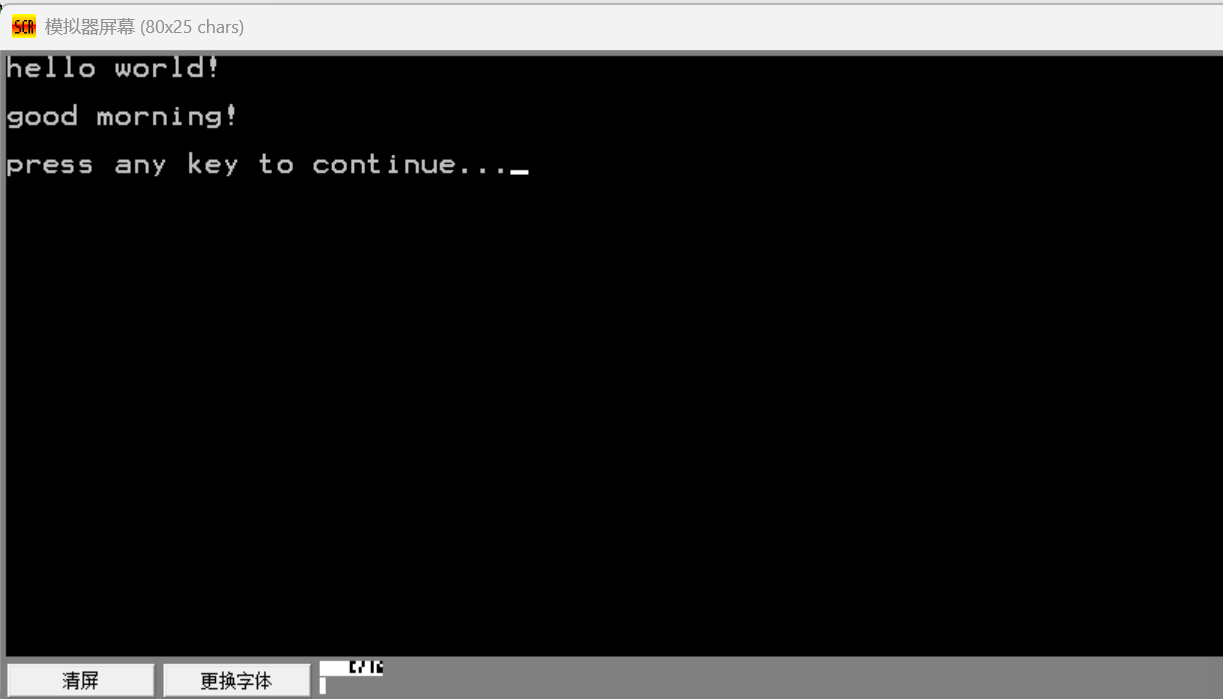
③ 字符输出逻辑错误导致显示不完整

问题表现： 若忘记检测字符串末尾的 0（即 \0 结束符），或 cmp 比较地址写错，将导致字符串无法完整输出或死循环。

解决方法： 使用 cmp byte ptr [bx], 0 检查字符是否结束，并及时跳转退出输出循环，确保 puts 子过程结束条件明确。

（2）程序输出结果

程序运行后，依次正确输出以下内容：



str1 与 str2 均成功显示，内容、格式、换行均符合预期；

提示语 prompt 通过 int 21h / ah=9 显示 $ 结尾字符串，行为符合 DOS 中断规范；

程序最后进入等待按键状态，使用 int 21h / ah=1 实现交互功能，验证了子过程调用和主流程控制逻辑的正确性；

所有字符串输出均包含空格、标点、换行符，显示字符个数也被准确统计，可在返回值 ax 中验证。

（3）现象分析与程序技巧总结

① 汇编输出字符串的标准流程体现清晰

通过子过程 puts 的编写，熟练掌握了如下操作序列：

使用 bx 传入字符串地址；

使用 cmp 检查 \0 结尾；

使用 int 21h / ah=2 实现逐字符输出；

使用 ax 返回输出字符计数。

该流程与 C 语言中的 puts() 函数逻辑等价，是汇编语言对高级函数的低层实现。

② 栈帧与寄存器管理清晰规范

子过程内部使用 push bp / mov bp, sp 建立帧结构；

使用 bp-2 位置存储局部变量 count，避免寄存器覆盖；

返回前完整恢复 bp 与 dx，使子过程符合 C 语言调用约定，支持嵌套调用与多次复用。

③ 程序结构清晰，标签逻辑合理

程序以 main 为主控入口，子过程 puts 独立封装。流程标签设计合理，结构如下：

loop\_entrance：字符处理主循环入口；

loop\_exit：字符串末尾检测跳出点；

利用循环配合 bx++ 与 dl 输出，形成稳定的逐字符显示控制机制。

1. 数组求和。

（1）调试情况与问题解决

在实际调试过程中，程序整体结构清晰，功能实现完备，实现了数组求和与结果输出的基本目标，但在开发初期也遇到以下几个典型问题：

① 参数压栈顺序错误导致子过程访问错位

问题表现：若在调用 arraysum 时压栈顺序颠倒，如先压 sum 地址，再压 array 地址，则在子过程中 [bp+6] 和 [bp+4] 所取地址发生错乱，导致结果错误或内存访问异常。

解决方法：严格遵守参数从右至左压栈的调用约定，即 push offset array 后再 push sum，使得 [bp+6] 正确对应数组地址。

② 栈帧未正确恢复导致主程序崩溃

问题表现：在 arraysum 或 print\_unsigned 等子过程中未完整恢复 bp 或 sp，或遗漏寄存器出栈，可能导致主程序返回后环境紊乱，引发错误输出或程序崩溃。

解决方法：在每个子过程中规范使用 push bp / mov bp, sp 建立帧结构，并在退出前依次恢复所有被压栈寄存器，确保调用现场完整恢复。

③ 十进制输出算法初期未处理零值

问题表现：当求和结果为 0 时，print\_unsigned 子过程将跳过输出环节，导致屏幕上无任何数字显示。

解决方法：在除法输出循环之前加入对 AX=0 的特判逻辑，输出单个字符 '0'，防止“零值不显示”的异常情况。

（2）程序输出结果

程序运行后，控制台输出结果如下：



分析说明：

程序正确输出提示字符串 "Sum: $"，说明 int 21h/ah=9 调用行为符合规范，输出终止于 $ 字符；

紧随其后的数值 150 即为数组 {10,20,30,40,50} 的累加结果，表明 arraysum 子过程逻辑正确，所有数组元素均被依次访问并加总；

输出格式为十进制形式，字符排列顺序从高位到低位，说明 print\_unsigned 中栈逆序输出机制工作正常；

最后一行提示 "press any key..." 正确显示，回车换行（ASCII 13/10）格式正确，用户按任意键后程序正常退出，验证主程序流程稳定。

（3）现象分析与程序技巧总结

① 汇编数组求和的地址操作技巧明确

使用 DI 寄存器作为数组指针，通过 [DI] 访问数组元素；

每次累加后 add DI, 2，因为 word 数据宽度为 2 字节，体现了对数据对齐和偏移控制的精准理解；

使用 CX 控制循环次数，由 length 常量初始化，避免硬编码，增强程序适应性。

② 十进制转换与输出策略高效规范

print\_unsigned 采用“除 10 取余”的标准算法，将每一位数字压栈，后续依次出栈实现从高位到低位输出；

使用 int 21h/ah=2 输出字符，辅以 '0' 偏移实现数值到 ASCII 字符的转换；

循环结构和计数器 CX 管理严格，确保输出位数精确，符合 C 语言 printf 效果。

③ 栈帧管理标准化，支持子过程复用与嵌套

每个子过程均规范建立栈帧push bp / mov bp, sp；

通过 [bp+6] 和 [bp+4] 访问传入参数，结构清晰，便于移植和扩展；

调用前压栈、调用后释放add sp, 4，确保栈空间不泄漏，体现了良好的程序设计规范。

3、汉诺塔问题。

（1）调试情况与问题解决

在程序调试过程中，整体功能逻辑清晰，模块结构分明，能够正确实现递归过程与格式化输出，但也遇到了若干典型问题和解决办法：

① 段寄存器初始化错误导致程序异常退出  
问题表现： 若未在主程序入口处正确初始化 mov ax, data 和 mov ds, ax，则字符串地址偏移将指向非法内存，导致格式化输出失败甚至程序崩溃。  
解决方法： 在 main 开始处严格初始化 ds 与 ss，并设置 sp 为堆栈顶端，确保各模块对数据段、栈段访问合法。

② 栈帧结构管理不当导致返回值错误或寄存器冲突  
问题表现： 若 hanoi 或 printf2 子过程未规范建立栈帧，或未在结束时恢复 bp 与通用寄存器，会造成调用嵌套时栈紊乱、参数访问错误、返回值 AX 被污染等问题。  
解决方法： 在所有子过程中采用统一结构：push bp → mov bp, sp → ... → mov sp, bp → pop bp，并确保保存/恢复所有被修改的寄存器，如 bx、cx、dx、ax。

③ 递归调用栈释放不完整导致 SP 累计偏移异常  
问题表现： 若 hanoi 子过程调用结束后未及时 add sp, 8 释放 4 个参数（2 字节×4），则 sp 会逐渐偏移栈底，引发子过程参数获取错误或程序崩溃。  
解决方法： 每次调用完 hanoi 或 printf2 后立即 add sp, 8 或 add sp, 4 回收对应参数区，保持栈结构稳定。

④ 格式化输出参数匹配错误  
问题表现： 在调用 printf2 时，若格式字符串与参数数量、顺序不对应，或 bp+偏移量 计算错误，可能导致错误字符输出或程序崩溃。  
解决方法： 使用 [bp-2] 管理格式化参数指针，依序读取 [bp+6]、[bp+8]、[bp+10]...，确保 %d、%c 格式与入栈参数严格一一对应。

（2）程序输出结果

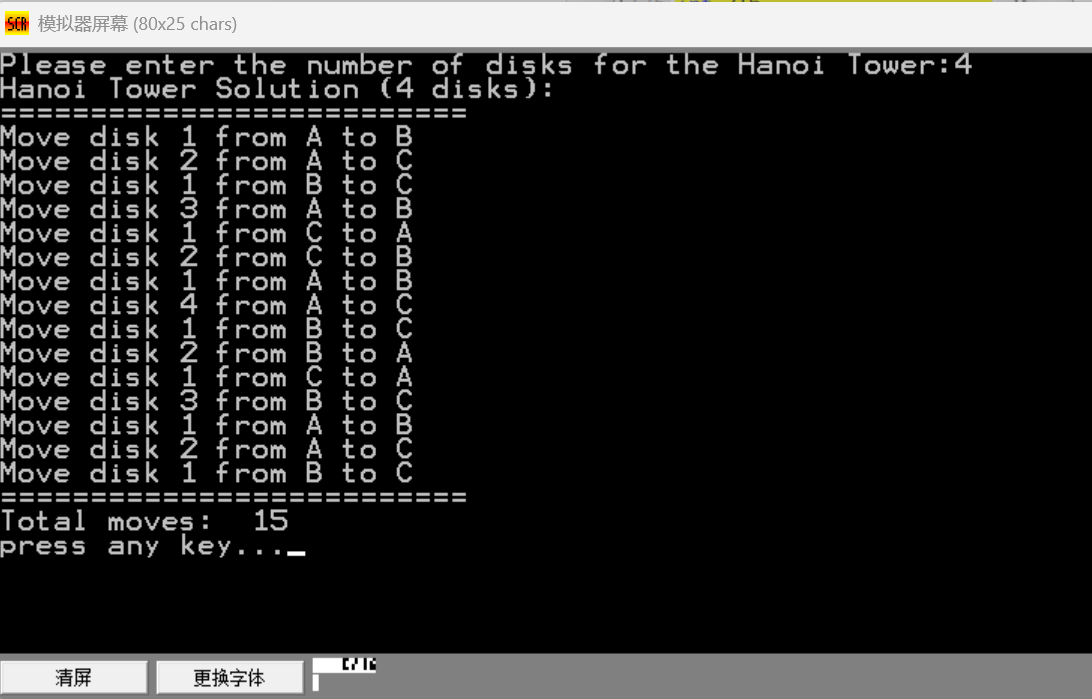
程序成功实现了完整的汉诺塔问题模拟，输出过程如下：

用户输入合法盘数，程序立即格式化输出：

输入3：

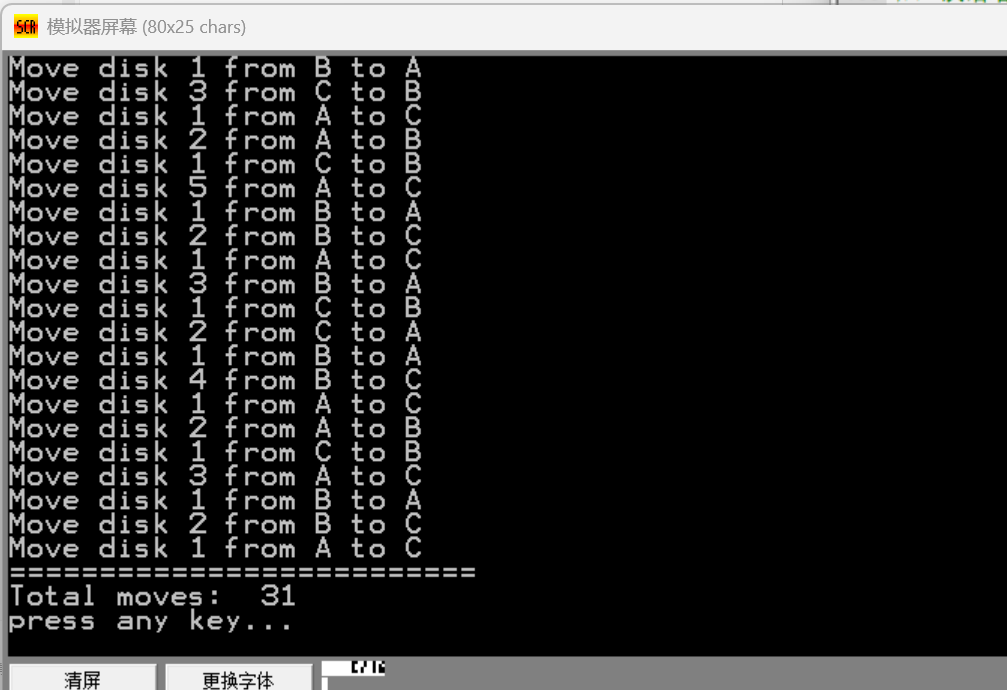


输入4：

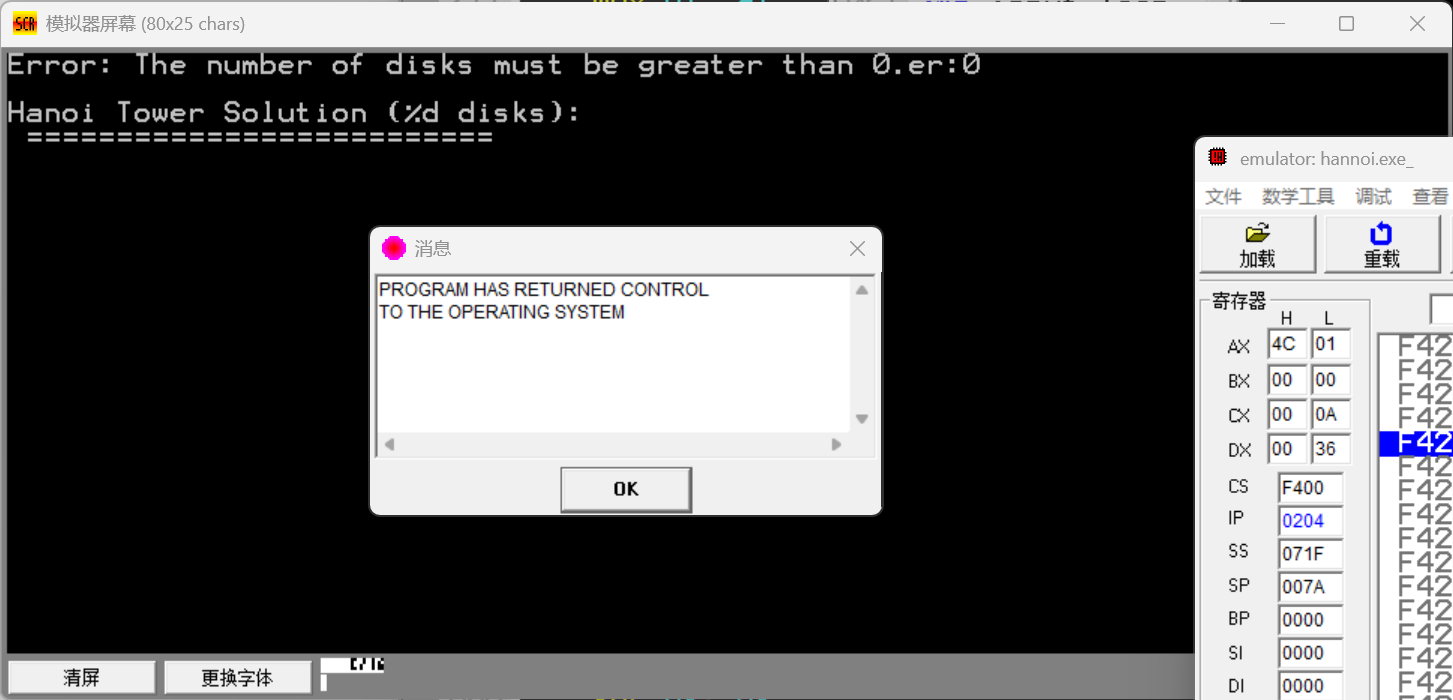


输入5：





输入非法数字如 0，会输出错误提示并优雅退出：



格式美观，逻辑正确，验证了主程序控制流、递归子程序与格式化输出功能的协同正确性。

（3）现象分析与程序技巧总结

① 递归结构在汇编中的实现方式  
该程序通过 call hanoi 与多层参数入栈，模拟了 C 语言中的递归调用过程。每次调用通过压栈传参，内部使用 [bp+offset] 取参，并在结束时 ret 回归上层调用，展示了基于栈帧的递归控制流设计模式。

② 格式化输出机制高度模块化  
printf2 子过程采用 %d/%c 占位符识别机制，通过 [bp+偏移量] 动态取参，组合 print\_unsigned 与 int 21h 实现灵活输出，是对 C 标准库 printf 函数的结构性还原，增强了汇编程序的可读性与扩展性。

③ 精确的栈帧管理保障程序稳定性  
每个子过程都严格维护调用现场：入栈所有修改寄存器，分配局部变量，退出时完整恢复，使得程序可以任意层数递归而不会造成栈混乱，是编写中大型汇编程序的核心技巧。

④ 输入解析与数值转换符合十进制逻辑  
通过多字符读取与 bx\*10 + digit 累加构造十进制数值，模拟 scanf("%d") 的整数输入逻辑，适配多位数字输入，提升程序交互能力。

4.实验心得体会  
（1）通过本次实验，我深刻体会到子程序在程序设计中的重要性。将功能模块化为独立的子程序，如 puts、arraysum 和 hanoi，不仅使代码更清晰易懂，还提高了复用性和可维护性。尤其是在汇编语言中，合理使用栈帧和参数传递机制可以有效避免寄存器冲突和数据污染。  
（2）调试过程中遇到了一些典型问题，比如栈帧未正确恢复导致主程序崩溃，或者参数压栈顺序错误引发访问错位。这些问题让我意识到调试不仅是检查语法错误，更是对程序逻辑和内存管理的深入理解。逐步排查并解决问题的过程极大地提升了我的调试能力。  
（3）汉诺塔问题让我真正理解了递归的思想及其在汇编语言中的实现方式。通过压栈传参和栈帧管理，程序能够优雅地处理多层递归调用。虽然递归在汇编中比高级语言复杂，但它依然是一种强大且高效的算法设计工具。  
（4）自定义实现类似 C 语言 printf 的功能是一项有趣的挑战。通过解析 %d 和 %c 格式符，并结合栈帧动态取参的方式，我们能够灵活输出各种数据类型。这让我更加理解高级语言标准库背后的工作机制。

（5）通过调试过程深刻认识到：子程序入口必须严格遵循PUSH BP + MOV BP, SP的栈帧建立流程。在汉诺塔递归案例中，任何寄存器,如AX/BX/CX/DX若未在子程序开头压栈保存，递归调用时必然导致关键数据被覆盖。这让我养成了在核心子程序中先保存现场PUSH寄存器组，再构建栈帧的编码习惯。