山东大学 计算机科学与技术 学院

汇编语言 课程实验报告

学号：202120130276	姓名：王云强	班级：21.2 班
实验题目：实验 7：示例 2.8		
实验学时：2	实验日期：2023.12.1	
实验目的： 1.掌握综合性汇编语言程序设计的方法。 2.掌握递归算法的设计与汇编表示。 3.全面回顾前述一切实验的内容。		
实验环境：Windows10、DOSBox-0.74、Masm64		
源程序清单： 1. HANOI.ASM（示例 2.8 源程序） 2. EX2_8.ASM（示例 2.8 修改后程序）		
编译及运行结果： HANOI.ASM（示例 2.8 源程序）编译结果：		

```
C:\>MASM HANOI
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [HANOI.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51642 + 448518 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

C:\>LINK HANOI

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [HANOI.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

HANOI.ASM（示例 2.8 源程序）运行结果：

```
C:\>HANOI
N=?
3

What is the name of spindle X?
A
What is the name of spindle Y?
B
What is the name of spindle Z?
C
A1C
A2B
C1B
A3C
B1A
B2C
A1C
```

```
What is the name of spindle X?
A
What is the name of spindle Y?
B
What is the name of spindle Z?
C
A1B
A2C
B1C
A3B
C1A
C2B
A1B
A4C
B1C
B2A
C1A
B3C
A1B
A2C
B1C
```

EX2_8.ASM（示例 2.8 修改后程序）编译结果：

```
C:\>masm EX2_8
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [EX2_8.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51602 + 448558 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

C:\>link EX2_8

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [EX2_8.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

EX2_8.ASM（示例 2.8 修改后程序）运行结果：

当两个文件都存在时：

```
C:\>EX2_8

    Please input input_filename: input.txt
    Please input output_filename: output.txt
    Successfully done!
```

input.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
3 X Y Z
```

output.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
X1Z
X2Y
Z1Y
X3Z
Y1X
Y2Z
X1Z
```

当输入文件不存在时：

```
C:\>EX2_8

    Please input input_filename: input.txt
    File not found!
```

当输出文件不存在时：

```
C:\>EX2_8

    Please input input_filename: input.txt
    Please input output_filename: output.txt
    File not found!
```

问题及收获：

1.在修改源代码之前，先对示例 2.8 源程序进行分析：
①主函数流程：
首先是读入数字 N（数字 N 可能有多位），所以读入字符串，然后调用 DECIBIN 子函数将读入的字符串转换成真实数字。之后读入 X、Y、Z 三个盘的名称（都是单字符内容），读入之后就调用 HANOI 子函数进行计算和输出。

②HANOI 子函数：
由于程序是递归程序，所以 HANOI 子函数是该程序的核心，这里主要讲一下怎么实现的递归，而不会着重讲如何计算汉诺塔的具体实现思路（因为这个思路比较简单且不是该节实验的重点，就是不断的向前递归，将 N 个盘子的内容考虑成为 N-1，最终到 1 个盘子，直接移动即可）。首先先判断 N 是否是 1，如果是 1，说明是递归的结尾，直接调用 PRINT 子函数输出即可。之后先调用 SAVE 子函数，保存起来 N（总盘子数）、X（X 轴盘子数）、Y（Y 轴盘子数）、 Z（Z 轴盘子数），之后让 BX-1（BX 储存的是 N 的值，即向前一步递归），交换 SI 和 DI 的值 （也就是 Y 轴和 Z 轴的盘子数）。之后调用 HANOI，进行递归调用。在递归调用结束后，需要恢复回来本次递归中的四个值（N、X、Y、Z），因此调用 RESTOR 子函数，将先前储存的 N、X、Y、Z 弹栈出来。再让 BX-1，交换 X 轴和 Y 轴盘子数，调用递归函数 HANOI。到这里计算完成，直接 RET 出来。总体子函数思路就是，先设立终止条件，若不符合终止条件，则进入函数执行。保存起来当前的重要变量所在的寄存器，之后进行向前递归，再将保存的寄存器弹栈出来。最后退出即可。

③PRINT 子函数：
函数的主要功能输出 CX 所代表的的盘子的名称和 N 的值和 DI 所代表的的盘子的名称，其中 N 的值由 BINIDEC 子函数来输出和计算。

④SAVE 子函数和 RESTOR 子函数 ：
主要功能就是保存四个值（N、X、Y、Z）所在寄存器，通过压栈和弹栈实现。

⑤DECIBIN 子函数：
其实之前实验中多次涉及这个函数，在此就简单说说。主要功能就是将字符串转换成真实数字。具体做法就是一个个读入字符，先判断是否是数字 ASCII，是的话就转换成数字，然后新读入数字时，之前得到的数字要乘 10（因为是从高位一直读到到低位）。

⑥CRLF 子函数：主要功能就是实现换行，然后从下一行开始输入/输出。

⑦BINIDEC 子函数：
主要功能是将 BX 中的数字，转换成真实数字进行输出。主要实现方式是，从 10000 开始进行除法，如果商不为 0，则将商显示出来，然后再依次除以 1000、100、10、1，最终将真实数字输出出来。

2.如果要用文件实现输入、输出，应该修改哪些地方？

在修改本源程序时，主要参考了示例 4.1、示例 4.2。首先是输入，在源程序中，是用输入字符串和字节进行的输入。要改成文件输入，可以参考示例 4.2，先写一个获取文件名（包括进行输入的文件名和进行输出的文件名）的子函数，之后再写一个打开文件的子函数，再写一个将文件中的内容读到全局变量数组中的子函数，再写一个将全局变量数组中的内容转换到对应位置的子函数（即将 N、X、Y、Z 从数组中分别读到 BX、CX、DI、SI）。这样就完成了将内容从文件转换到与源程序匹配的寄存器中的输入过程了。

参考示例 4.2，获取文件名及内容函数如下：

```
;-------------------------------------------------------------------------
GETING          PROC            NEAR                            ;获取文件名
                PUSH            AX
                PUSH            BX
                PUSH            CX
                PUSH            DX

                MOV             DX, OFFSET MESS_GETNAME
                MOV             AH, 09H
                INT             21H

                MOV             DX, OFFSET BUF_SIZE
                MOV             AH, 0AH
                INT             21H

                MOV             BL, S_BUF
                MOV             BH, 0
                MOV             NAMES[BX], 0                    ;insert 0 to form the ascii string
NAME_MOVE:
                DEC             BX
                MOV             AL, BUF[BX]
                MOV             NAMES[BX], AL                   ;move the line got into name string
                JNZ             NAME_MOVE

                POP             DX
                POP             CX
                POP             BX
                POP             AX
                RET
GETING          ENDP
;-------------------------------------------------------------------------
OPENF           PROC            NEAR                            ;打开文件
```

```
;-------------------------------------------------------------------------
OPENF           PROC            NEAR                            ;打开文件
                PUSH            BX
                PUSH            CX
                PUSH            DX
                MOV             DX, OFFSET NAMES
                MOV             AL, 0                           ;打开NAMES对应的文件
                MOV             AH, 3DH
                INT             21H
                MOV             HANDLE, AX
                MOV             AX, 1
                JNC             QUTING
                MOV             AX, 0
QUTING:
                POP             DX
                POP             CX
                POP             BX
                RET
OPENF           ENDP
;-------------------------------------------------------------------------
GET_NUM         PROC            NEAR                            ;将文件内容读到BUF中
                PUSH            BX
                PUSH            CX
                PUSH            DX
                MOV             CX, 200
                MOV             BX, HANDLE
                MOV             DX, OFFSET BUF
                MOV             AH, 3FH
                INT             21H                             ;读到BUF数组中
                MOV             AX, 1
BACK:
                POP             DX
                POP             CX
                POP             BX
                RET
```

之后是输出，在源程序中，输出是直接输出到屏幕上的。在这里，需要首先打开输出文件，然后将对应内容通过 21H 号中断的 40H 号内容输出到文件中，主要修改的是 PRINT 子函数以及 BINIDEC 子函数。

修改如下：

```
;-------------------------------------------------------------------------
PRINT           PROC            NEAR
                PUSH            BX
                MOV             BX, HANDLE2
                MOV             DX, CX                          ;PRINT X
                MOV             OUTING, DX                      ;outing为DX的值，输出一位
                LEA             DX, OUTING
                PUSH            CX                              ;暂时储存CX
                MOV             CX, 1
                MOV             AH, 40H
                INT             21H
                POP             CX

                POP             BX
                CALL            BINIDEC                         ;PRINT N

                MOV             DX, DI                          ;PRINT Z
                MOV             OUTING, DX                      ;outing为DX的值，输出一位
                LEA             DX, OUTING
                PUSH            CX                              ;暂时储存CX
                PUSH            BX
                MOV             BX, HANDLE2
                MOV             CX, 1
                MOV             AH, 40H
                INT             21H
                POP             BX
                POP             CX
                CALL            CRLF                            ;skip to next line
                RET
PRINT           ENDP
;-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------
BINIDEC         PROC            NEAR
                                                        ;procedure to convert binary number in BX to decimal
                                                        ; on console screen
                PUSH            BX
                PUSH            CX
                PUSH            SI
                PUSH            DI
                MOV             FLAG, 0
                MOV             CX, 5
                LEA             SI, CONSTANT
DEC_DIV:
                MOV             AX, BX                  ;number high half
                MOV             DX, 0                   ;zero out low half
                DIV             WORD PTR [SI]           ;divide by contant
                MOV             BX, DX                  ;remainder into BX
                MOV             DL, AL                  ;quotient into DL

                CMP             FLAG, 0                 ;have not leading zero
                JNZ             PRINT1
                CMP             DL, 0
                JE              SKIP
                MOV             FLAG, 1
                                                        ;PRINT the contents of DL on screent
PRINT1:
                ADD             DL, 30H                 ;convert to ASCII
                MOV             DH, 0
                MOV             OUTING, DX
                LEA             DX, OUTING
                PUSH            CX
                PUSH            BX
                MOV             CX, 1
                MOV             BX, HANDLE2
                MOV             AH, 40H                 ;display function
                INT             21H
```

3.关于子程序入参、传参的三种方式，结合实践，是如何看待和评价的？
首先先评价一下寄存器传参，总体使用起来感觉还可以，麻烦的地方在于需要注意什么时候需要保存起来，需要保存哪几个寄存器，什么时候再恢复回来，需要恢复哪几个寄存器。不过个人认为好处在于，在实践中传参比较方便（毕竟是现成的），而且有 PUSH 和 POP 指令进行辅助保护和恢复，总体评价还是不错的（但是像这个实验一样，需要一直占用寄存器，还需要花费一些精力去小心一些存在寄存器中的值被覆盖，毕竟寄存器太少了，也算是一个不足之处）。

再评价一下全局变量传参（之前实验和这次实验都用到一些），个人感觉挺好用的，就是需要提前在数据段进行声明，一般麻烦，好处在于不用考虑寄存器覆盖等问题，基本上全局变量来传参的话，不太用考虑被覆盖的问题，坏处在于比较占用空间。

最后评价一下用栈传递参数，由于没有实践过，所以只能从理论知识层面上进行分析，根据课堂上所讲例子和例题来看，这个要比寄存器传参更需要专注，因为一不小心就错把栈中这一个元素当成下一个元素进行使用了，容易出问题，但是如果熟练的话（加上大脑清晰），应该是比较好的传参方式，而且不用实现定义，也不用担心寄存器覆盖，通过一个个指针就可以找到对应内容，挺好的方式，就是比较考验熟练度。从理论层面上讲，最好的肯定是栈传递参数，可存参数多，速度也比较快。寄存器传 参主要问题是寄存器太少，用多了，就会频繁的需要压栈、出栈来保护寄存器。全局变量 是最简单、但是比较不好的方式，因为速度比较慢，且要额外显式地占用一些空间。