华 中 科 技 大 学

课 程 实 验 报 告

课程名称：汇编语言程序设计实验

实验时间：2022年 月 日

实验地点：

指导教师：

专业班级：

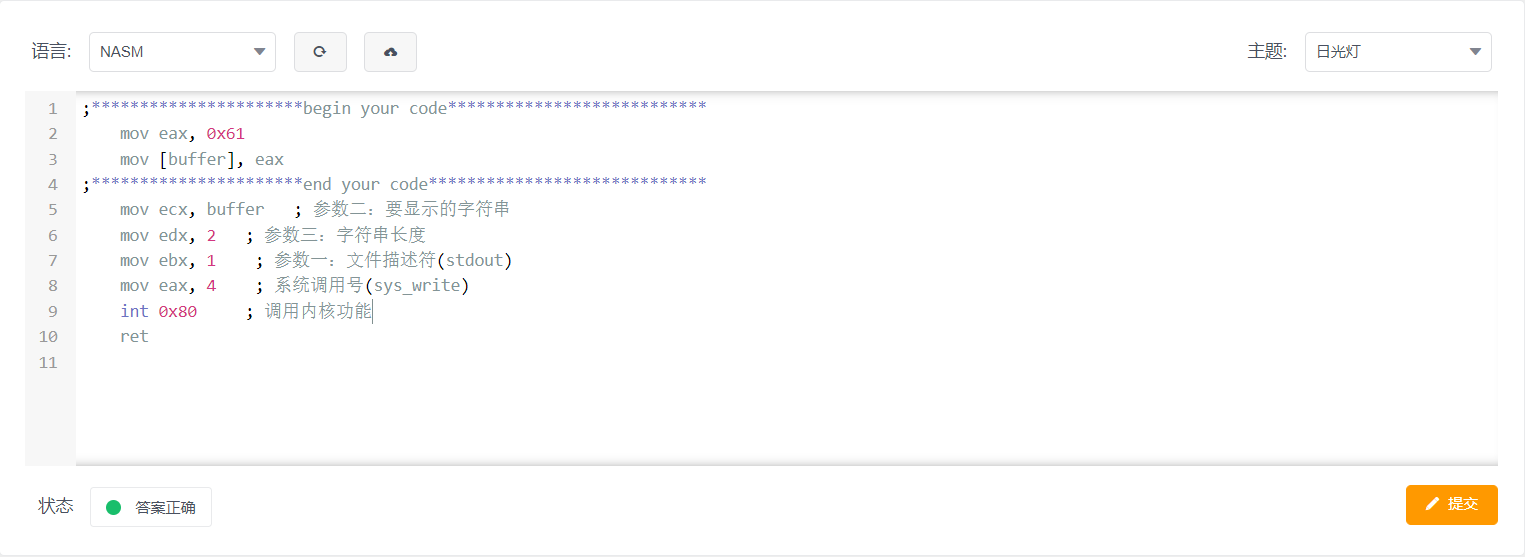
学 号：

姓 名：

报告日期：

实验一代码与实验截图

实验1.1



1. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*begin your code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. mov eax, 0x61
3. mov [buffer], eax
4. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end your code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
5. mov ecx, buffer   ; 参数二：要显示的字符串
6. mov edx, 2   ; 参数三：字符串长度
7. mov ebx, 1    ; 参数一：文件描述符(stdout)
8. mov eax, 4    ; 系统调用号(sys\_write)
9. **int** 0x80     ; 调用内核功能
10. ret

实验1.2



1. mov  eax,[x]    ; 让eax的值等于x，eax=x
2. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*begin your code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
3. sub eax, [y]
4. mov ebx, [z]
5. imul ebx
6. add eax, 32h
7. mov edx, 0h
8. mov ebx, [v]
9. idiv ebx
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*you’re your code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
11. call print\_eax    ;通过打印eax函数输出此时的商
12. mov  eax ,edx     ;将存放在edx中的余数值传给eax
13. call print\_eax    ;通过打印eax函数输出此时的余数
14. ret

实验二代码与实验截图

实验2.1

1. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*begin your code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. ; msg 对应data段中一个字符串首地址，用于linux下系统调用的输出参数
3. ; len 对应字符串长度，用于linux下系统调用的输出参数
4. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Begin your code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
5. mov eax, 'ever'
6. mov [msg+6], eax
7. mov eax, 'yone'
8. mov [msg+10], eax
9. mov eax, '!'
10. mov [msg+14], eax
11. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End your code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
12. mov ecx, msg   ; 参数二：要显示的字符串
13. mov edx, len   ; 参数三：字符串长度
14. mov ebx, 1    ; 参数一：文件描述符(stdout)
15. mov eax, 4    ; 系统调用号(sys\_write)
16. **int** 0x80     ; 调用内核功能
17. ret

实验2.2

1. ;section .data
2. ;flag db 0
3. ;v\_word dw 'hU' ,'ST'    ; data段，此处修改无效
4. ;target times 10 db 0
5. ;crlf db 0dh ,0ah
6. ;len equ $-v\_word
7. section .txt      ; 代码段声明，下列代码将放入代码段中
8. global \_start      ; 指定入口函数，global修饰是为了让外部可以引用
9. \_start:
10. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*input your code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
11. mov ax, 'HU'
12. mov [v\_word], ax
13. mov eax, '70th'
14. mov [target], eax
15. mov ecx, 6
16. mov esi, 4
17. mov al, '!'
18. again:
19. mov [target+esi], al
20. inc esi
21. loop again
22. mov al, 0h
23. mov [crlf], al
24. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* end  your code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
25. mov ecx, v\_word   ; 参数二：要显示的字符串
26. mov edx, len  ; 参数三：字符串长度
27. mov ebx, 1    ; 参数一：文件描述符(stdout)
28. mov eax, 4    ; 系统调用号(sys\_write)
29. **int** 0x80     ; 调用内核功能
30. mov ebx,0;参数为0，exit(0)
31. mov eax,1;系统调用号为1，sys\_exit
32. **int** 0x80

实验三代码与实验截图

实验3.1

1. ;数据段中的存储信息
2. ;section .data
3. ;string db "?????????????????"
4. ;slength equ $-string
5. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*begin your code here\*\*\*\*\*\*\*\*\*\*\*\*\*\*
6. ;string中存放处理后的结果
7. ;edx寄存器存放处理后的字符串长度
8. mov ebx, 0
9. mov esi, 0
10. mov ecx, slength
11. again:
12. mov al, string[esi]
13. cmp al, 41h ;判断是否为字母
14. js num\_space    ;不是字母跳转
15. mov string[ebx], al     ;是字母字符前移
16. inc ebx
17. num\_space:
18. mov al, 0h
19. mov string[esi], al
20. inc esi
21. loop again  ;遍历
22. mov edx, ebx
23. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*end your code here\*\*\*\*\*\*\*\*\*\*\*\*\*\*
24. ;后续程序内容，此部分不用实现
25. ;call prinf\_string ;;调用参数string,edx打印字符串
26. ;exit

实验3.2

1. ;数据段中的定义数据
2. ;section .data
3. ;string db "?????????????????"
4. ;slength equ $-string
5. ;cmpstr db 'ABC'
6. ;cmpslength equ $-cmpstr
7. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*begin your code here\*\*\*\*\*\*\*\*\*\*\*\*\*\*
8. ;输出结果存放在eax寄存器中
9. mov esi, 0
10. mov ecx, slength
11. upper\_again1:   ;将字符串string大写
12. mov al, string[esi]
13. cmp al, 61h
14. js upper1   ;将小写字母替换为大写
15. sub al, 20h
16. upper1:
17. mov string[esi], al
18. inc esi
19. loop upper\_again1
20. mov esi, 0  ;指针指向string
21. mov edx, 0  ;指针指向cmpstr
22. mov eax, 0  ;记录次数
23. mov ecx, slength
24. loop\_again: ;遍历字符串
25. push esi
26. loop\_again1:
27. mov bl, string[esi]
28. cmp bl, cmpstr[edx]
29. jnz nequ    ;字符不相同
30. inc edx
31. inc esi
32. cmp edx, cmpslength ;cmpstr是否到达末尾
33. js loop\_again1
34. inc eax ;次数增加1
35. nequ:
36. pop esi
37. mov edx, 0
38. inc esi
39. loop loop\_again
40. ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*end your code here\*\*\*\*\*\*\*\*\*\*\*\*\*\*
41. ;后续程序输出eax中的数值，不需要实现
42. ;call print\_eax
43. ;jmp quit

实验四：汇编语言子程序设计与C语言混合编程

1. 实验目的
2. 掌握汇编语言子程序设计方法；
3. 熟悉汇编语言主程序和子程序不同的参数传递方法;
4. 了解C语言和汇编语言混合编程方法以及主、子程序之间参数传递的机制。
5. 实验内容

设计汇编子程序来实现斐波那契数列计算，输入项数n，输出对应第n项的数值。

斐波那契数列（Fibonacci sequence），又称黄金分割数列，因数学家莱昂纳多·斐波那契（Leonardo Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8、……。

斐波那契数列以如下被以递推的方法定义：

F(0)=0，F(1)=1,F(n)=F(n - 1)+F(n - 2)（n≥ 2，n∈ N\*）

**注意：由于宽度限制，不考虑溢出的情况，可实现n<20的斐波那契数列数列计算即可。**

1. 实验要求
2. 任务1、2、3需要首先在oj平台上提交通过，在实验报告中需要填写可masm6.x以上版本编译通过简化段定义的完整汇编语言程序包括输入输出，**不只是子程序代码片段**。
3. 理解思考任务1、2、3中不同参数传递方式对汇编子程序编写的区别，掌握dosbox下调试汇编程序的方法。
4. 任务4中在不同的C语言开发环境中实现与汇编语言程序的混合编程，其操作方法有可能是不同的。请大家选择白己熟悉的C语言开发环境并查找相关的资料完成本实验。在实验报告中，详细地描述采用的开发环境及其实现方法。
5. 观察C语言编译器中对各种符号的命名规则（指编译器内部可以识别的命名规则，比如，符号名前面是否加下划线“\_”等），主、子程序之间参数传递的机制，通过栈传递参数后堆栈空间回收的方法。
6. 对混合编程形成的执行程序，用调试工具观察由C话言形成的程序代码与由汇编语言形成的程序代码之间的相互关系，包括段、偏移的值，汇编指令访问C的变量时是如何翻译的等。
7. 通过本次实验，希望大家掌握汇编子程序的设计方法以及不同的编程语言是可以协同解决一个问题的，而且可以利用不同语言的特点来更好地解决问题；利用汇编语言的知识，能够更好地理解高级语言的内部处理原理与策略。
8. 实验过程

**一、实验分析**

实验要求使用递归方式实现斐波拉契数列的计算，当n ≤ 2 时，函数返回1，此时不需要递归计算，当n > 2时，第n项结果为前两项结果之和，即要求feibo(n)，需要计算feibo(n - 1)和feibo(n - 2)，返回结果为feibo(n) = feibo(n-1)+feibo(n-2)。计算流程如图：

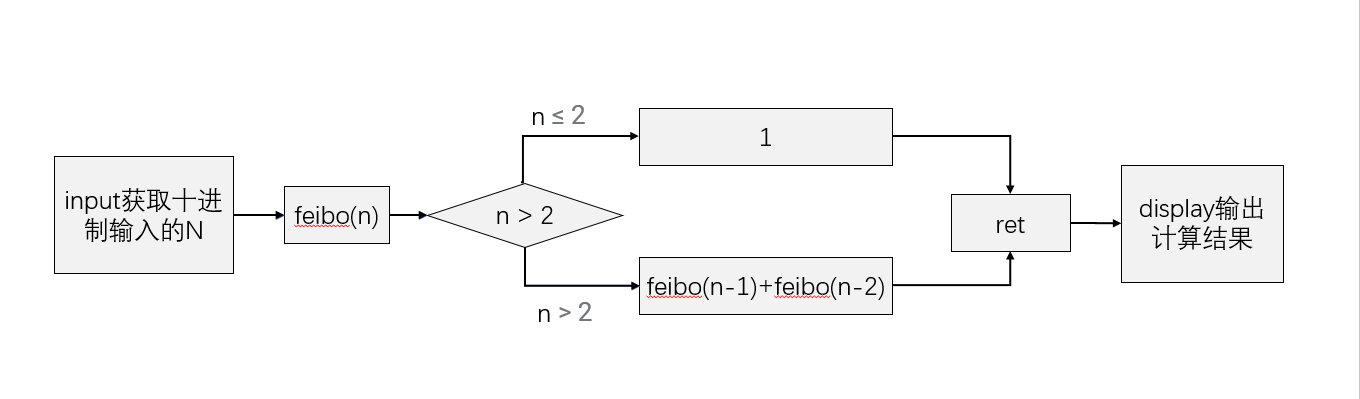


图1. 程序流程图

程序要求通过dos系统调用以十进制形式输入和输出参数，输入由子程序input实现，输出由子程序display实现。

input子程序使用01h功能号调用dos字符输入，每次读取一位十进制数字的ASCII码，将ASCII码转换为对应的数，将已输入的数乘10再与此次输入的数相加，直至获取到非数字字符的输入，最终得到输入的十进制数，并将其返回。

display子程序使用02h功能号调用dos字符输出，通过将n除以10取余的方式逐一得到n的每一个十进制位上的数，此时得到的每一位排序方式时由低位到高位，而输出时需要按高位到低位输出，因此需要逆序排列，此处使用堆栈后进先出的特点，计算时将低位到高位依次压入栈中，输出时按高位到低位的顺序依次取出，将其转换为对应的ASCII码，传入dl寄存器，调用02h功能号进行输出。

input的汇编实现：

1. input proc          ; 十进制输入，出口参数在dx中
2. mov dx, 0
3. ag:
4. mov ah, 01h     ; 调用dos功能输入字符
5. **int** 21h
6. cmp al, 30h     ; 输入字符不为数字，结束输入
7. jl endi
8. cmp al, 39h     ; 输入字符不为数字，结束输入
9. jg endi
10. sub al, 30h     ; 将ACSCII转化为十进制
11. xchg al, dl
12. mov bl, 10
13. imul bl
14. add dl, al
15. jmp ag
16. endi:
17. mov bx, dx
18. mov ah, 02h
19. mov dl, 0dh     ; 换行
20. **int** 21h
21. mov dx, bx
22. ret
23. input endp

display的汇编实现

1. display proc        ; 十进制输出，入口参数在dx中
2. mov ax, dx
3. mov cx, 0
4. again:
5. cmp ax, 0
6. jz enddisp
7. mov dx, 0
8. mov bx, 10
9. idiv bx ; 对10取余，逐位储存在堆栈中
10. push dx ; 将余数压入堆栈，使其逆置
11. inc cx
12. jmp again
13. enddisp:
14. l:
15. mov ah, 02h
16. pop dx  ; 逐次取出余数
17. add dl, 30h ; 转换为ASCII码
18. **int** 21h
19. loop l
20. ret
21. display endp

## 任务1：计算斐波那契数列(用寄存器传递参数)

请编写vscode环境下完整的MASM可编译通过的汇编程序，其中需要使用寄存器eax传递入口和出口参数的形式递归子程序feibo1来计算斐波那契数列。输入通过dos系统调用获取，输出需要通过dos系统调用输出显示为10进制数。

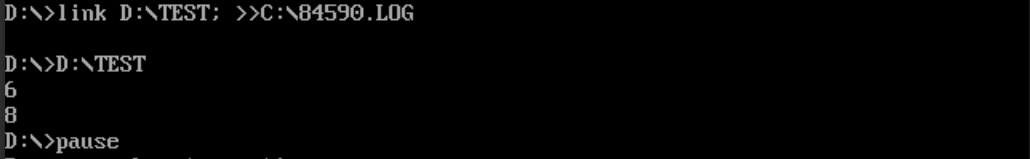
feibo1子程序用于计算斐波拉契数列，使用ax寄存器传递入口参数n和出口参数。计算过程由递归实现feibo1(n)=feibo1(n-1)+feibo1(n-2)

input子程序用于调用dos系统输入10进制数，出口参数在dx寄存器中

display子程序用于调用dos系统输出10进制数，入口参数在dx寄存器中，将10进制数的每一位提取后用堆栈倒序取出，调用02H功能号逐个字符输出

1. .model small
2. .stack
3. .data
4. .code
5. .startup
6. call input      ; 调用input子程序获取十进制输入的n，出口参数保存在dx寄存器中
7. mov ax, dx      ; 将入口参数n传入寄存器ax
8. call feibo1
9. mov dx, ax      ; 从ax中取出计算结果
10. call display    ; 十进制输出，入口参数在dx中
11. .exit 0
12. feibo1 proc         ; 入口参数n使用ax寄存器传递，出口参数result使用寄存器ax传递
13. push bp
14. mov bp, sp
15. push bx
16. push dx
17. cmp ax, 2
18. jg  r           ;n>2时递归计算
19. mov ax, 1
20. jmp endf        ;n<=2时返回1
21. r:
22. mov bx, ax
23. sub ax, 1
24. call feibo1     ;feibo1(n-1)，入口、出口参数在ax中
25. mov dx, ax
26. mov ax, bx
27. sub ax, 2
28. call feibo1     ;feibo1(n-2)，入口、出口参数在ax中
29. add ax, dx      ;feibo1(n-1) + feibo1(n-2)
30. endf:
31. pop dx
32. pop bx
33. pop bp
34. ret
35. feibo1 endp
36. end

运行结果：



## 任务2：计算斐波那契数列(用栈传递参数)

请编写vscode环境下完整的MASM可编译通过的汇编程序，其中需要使用堆栈传递入口参数和出口参数形式递归子程序feibo2来计算斐波那契数列。输入通过dos系统调用获取，输出需要通过dos系统调用输出显示为10进制数。

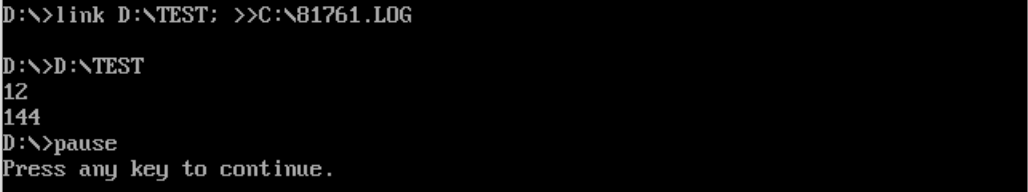
feibo2子程序用于计算斐波拉契数列，使用堆栈传递入口参数n和出口参数。计算过程由递归实现feibo2(n)=feibo2(n-1)+feibo2(n-2)

input子程序用于调用dos系统输入10进制数，出口参数在dx寄存器中

display子程序用于调用dos系统输出10进制数，入口参数在dx寄存器中，将10进制数的每一位提取后用堆栈倒序取出，调用02H功能号逐个字符输出

1. .model small
2. .stack
3. .data
4. .code
5. .startup
6. call input      ; 调用input子程序获取十进制输入的n，出口参数保存在dx寄存器中
7. push dx         ; 将入口参数压入栈中
8. call feibo2
9. pop ax          ; 从栈中取出计算结果
10. mov dx, ax
11. call display
12. .exit 0
13. feibo2 proc
14. push bp
15. mov bp, sp
16. push ax
17. push bx
18. push dx
19. mov ax, [bp+4]  ;从栈中取出n
20. cmp ax, 2
21. jg  r           ;n>2时递归计算
22. mov ax, 1
23. jmp endf        ;n<=2时返回1
24. r:
25. mov bx, ax
26. sub ax, 1
27. push ax         ;向栈中压入n-1
28. call feibo2     ;feibo2(n-1)
29. pop ax          ;从栈中取出n-1的结果
30. mov dx, ax
31. mov ax, bx
32. sub ax, 2
33. push ax         ;向栈中压入n-2
34. call feibo2     ;feibo2(n-2)
35. pop ax          ;从栈中取出n-2的结果
36. add ax, dx      ;feibo2(n-1) + feibo2(n-2)
37. endf:
38. mov [bp+4], ax  ;将计算结果放入栈中返回
39. pop dx
40. pop bx
41. pop ax
42. pop bp
43. ret
44. feibo2 endp
45. end

运行结果：



## 任务3：计算斐波那契数列(用公共变量传递参数)

请编写vscode环境下完整的MASM可编译通过的汇编程序，其中需要使用N作为入口参数和result作为出口参数的形式子程序feibo3来计算斐波那契数列。输入通过dos系统调用获取，输出需要通过dos系统调用输出显示为10进制数。

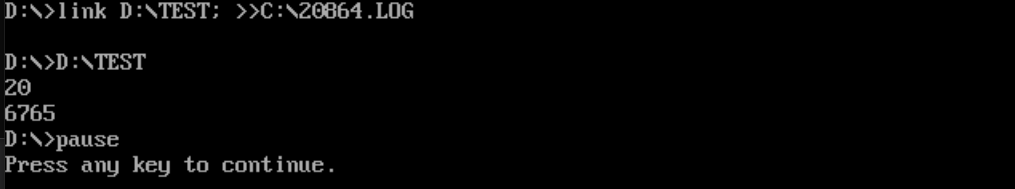
feibo3子程序用于计算斐波拉契数列，使用变量N传递入口参数n和使用变量result传递出口参数。计算过程由递归实现feibo3(n)=feibo3(n-1)+feibo3(n-2)

input子程序用于调用dos系统输入10进制数，出口参数在dx寄存器中

display子程序用于调用dos系统输出10进制数，入口参数在dx寄存器中，将10进制数的每一位提取后用堆栈倒序取出，调用02H功能号逐个字符输出

1. .model small
2. .stack
3. .data
4. N dw 0
5. result dw 0
6. .code
7. .startup
8. call input          ; 调用input子程序获取十进制输入的n，出口参数保存在dx寄存器中
9. mov [N], dx         ; 将入口参数传送到变量N中
10. call feibo3
11. mov dx, [result]    ; 将计算结果从result中取出
12. call display        ; 十进制方式显示
13. .exit 0
14. feibo3 proc
15. push bp
16. mov bp, sp
17. push ax
18. push bx
19. push dx
20. mov ax, [N]         ;从变量N中取出n
21. cmp ax, 2
22. jg  r               ;n>2时递归计算
23. mov ax, 1
24. jmp endf            ;n<=2时返回1
25. r:
26. mov bx, ax
27. sub ax, 1
28. mov [N], ax         ;将n-1送入变量N中
29. call feibo3         ;feibo3(n-1)
30. mov dx, [result]    ;从栈中取出n-1的结果
31. mov ax, bx
32. sub ax, 2
33. mov [N], ax         ;将n-2送入变量N中
34. call feibo3         ;feibo3(n-2)
35. mov ax, [result]    ;从栈中取出n-2的结果
36. add ax, dx          ;feibo3(n-1) + feibo3(n-2)
37. endf:
38. mov [result], ax    ;将计算结果放入result中返回
39. pop dx
40. pop bx
41. pop ax
42. pop bp
43. ret
44. feibo3 endp
45. end

运行结果：



## 任务4：计算斐波那契数列(混合编程)

请用汇编语言编写计算斐波那契数列数列的子程序feibo4，用C语言函数来获取输入、调用feibo4汇编函数，并且显示子程序返回的结果。

**1、实验环境**

操作系统：wsl2 Ubuntu 20.04

编译器版本：GCC version 11.1.0 (Ubuntu 11.1.0-1ubuntu1~20.04)

调试工具：IDA Pro 7.5 sp3

**2、实验步骤**

1. 了解C语言内联汇编的特点
2. 了解C语言函数调用入口参数和出口参数的传递方式
3. 编辑、调试内联汇编程序
4. 分析不同平台函数调用差异

**3、实验过程**

1. 了解C语言内联汇编的特点

GCC支持在C/C++代码中嵌入汇编代码，这些汇编代码被称作是”GCC Inline ASM”（GCC内联汇编）；

GCC中的基本的内联汇编格式为：

asm [volatile] ("instruction list":Output:Input);

asm用来声明一个内联汇编表达式；volatile是可选选项，使用该关键字可以关闭GCC对汇编指令的优化，以便在编译后保留每一条汇编指令；

”instruction list”为汇编指令列表，本学期所学8086汇编为Intel风格的汇编语句，其格式为：【[名称[:]] 指令码 目的操作数DST,源操作数SRC ;注释】，GCC编译器的内联汇编为AT&T风格的汇编语句，其格式为【[名称[:]] 指令码 源操作数SRC,目的操作数DST ;注释】，除源操作数SRC和目的操作数DST的位置不同外，两种风格所含指令和指令操作基本相同。

OUTPUT和INPUT部分用来声明内联汇编的输出、输入参数，使用“:”进行分隔，其声明格式为”=m/a/b/c/d/r”(src)，“=”表示可读，不使用“=”，OUTPUT声明的参数仅支持写操作，INPUT部分不能使用“=”，仅支持读操作，；“m”表示使用内存地址传递参数、“a/b/c/d/r”表示使用寄存器传递参数，使用“a”表示从RAX、EAX、AX、AL寄存器中选择一个寄存器，“r”表示从RAX、RBX、RCX、RDX、EAX…等寄存器中选择一个寄存器；src为参数，INPUT部分声明的参数会在执行内联汇编语句前被传送到指定位置（寄存器或内存），OUTPUT部分声明的参数会在执行内联汇编语句后被从指定位置传送到变量中，在内联汇编语句中通过%i（i = 0、1、2、3…）调用第i个参数。

内联汇编中使用%%eax的形式表示寄存器，%0的形式表示参数、$1的形式表示立即数。

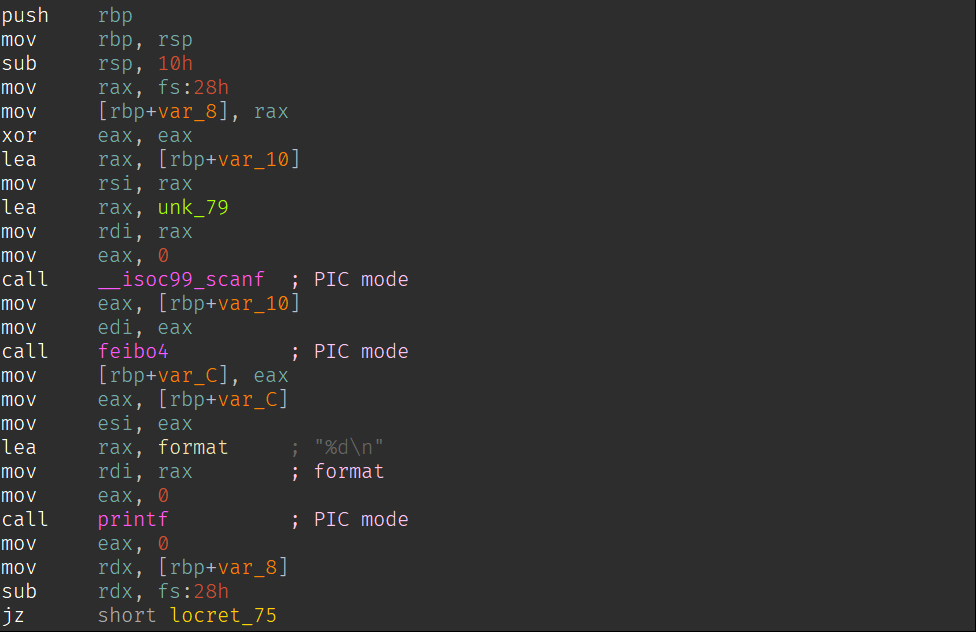
1. 了解C语言函数调用入口参数和出口参数的传递方式

使用递归方法计算斐波拉契数列，需要在函数中调用函数，因此在内联汇编部分需要调用C语言的函数，在调用feibo4函数前，需要将入口参数传入指定寄存器，函数运行结束后，还需要从指定位置取出出口参数，所以还需要找到feibo4函数的入口参数以及出口参数的传递方式，才能实现内联汇编和C语言函数的相互调用。

编写如下源文件main.c，使用scanf从标准输入获取n，即feibo4函数的入口参数，调用feibo4计算第n个斐波拉契数，计算后使用printf函数标准输出。

1. #include <stdio.h>
3. **int** feibo4(**int** n);
5. **int** main()
6. {
7. **int** n, result;
8. scanf("%d", &n);
9. result = feibo4(n);
10. printf("%d\n", result);
11. **return** 0;
12. }

执行编译命令gcc -c main.c -o main.o，将编译后的输出文件main.o使用IDA进行反编译。



调用scanf函数后，从标准输入得到的n被传送入eax寄存器，调用feibo4函数前，mov edi, eax命令将eax中的n移入edi寄存器后，再调用feibo4函数，因此edi为feibo4函数的入口参数寄存器；再调用feibo4函数后，mov [rbp+var\_C], eax表示将eax中的值取出存放到内存变量中，因此可以得知eax为feibo4函数的出口参数寄存器。

从上述实验中可以得知，在内联汇编部分调用feibo4函数时，需将入口参数n传入edi寄存器，计算结果保存在eax寄存器中。

1. 编辑、调试内联汇编程序

如下为使用内联汇编实现的feibo4函数，保存在feibo4.c源文件中

1. **int** feibo4(**int** n)           // 入口参数edi，出口参数eax
2. {
3. **int** result;
4. asm **volatile**(           // 内联汇编部分，入口参数n使用内存指针n传递，出口参数使用内存指针result传递
5. "mov %1, %%eax;"    // 从指向n的内存地址取出n的值传入eax中
6. "cmp $2, %%eax;"    // 比较n与2的大小
7. "jg r;"             // n大于2时递归计算
8. "mov $1, %%eax;"    // n小于等于2时直接返回1
9. "jmp endf;"
10. "r:;"
11. "sub $1, %%eax;"    // n - 1
12. "mov %%eax, %%edi;" // 将n - 1传递给edi寄存器，函数入口参数
13. "call feibo4;"      // feibo4(n-1)
14. "mov %%eax, %0;"    // 将函数出口参数保存到result所在的内存中
15. "mov %1, %%eax;"    // 从指向n的内存地址取出n的值传入eax中
16. "sub $2, %%eax;"    // n - 2
17. "mov %%eax, %%edi;" // 将n - 2传递给edi寄存器，函数入口参数
18. "call feibo4;"      // feibo4(n-2)
19. "add %0, %%eax;"    // %0中储存的为feibo4(n-1)的值，将其加到feibo4(n-2)的计算结果中
20. // 即eax中储存的为feibo4(n-1)+feibo4(n-2)
21. "endf:"
22. "mov %%eax, %0;"    // 将返回值传送到result所在的内存中
23. : "=m"(result)      // OUTPUT出口参数，使用内存传参，通过%0调用
24. : "m"(n));          // INPUT入口参数，使用内存传参，通过%1调用
25. **return** result;
26. }

使用GCC编译器进行编译连接，生成可执行文件a.out

1. ❯ gcc -c main.c -o main.o
2. ❯ gcc -c feibo4.c -o feibo4.o
3. ❯ gcc main.o feibo4.o -o a.out
4. ❯ ./a.out
5. 7
6. 13
7. ❯ ./a.out
8. 9
9. 34
10. ❯ ./a.out
11. 20
12. 6765

输入多个测试数据，n = 7时，计算结果为13，n = 9时，计算结果为34，n = 20时，计算结果为6765，计算结果均正确。

1. 分析不同平台函数调用差异

在windows系统下编译运行

操作系统：Windows 10 专业版21H2

编译器版本：gcc version 12.2.0 (x86\_64-posix-seh-rev0, Built by MinGW-W64 project)

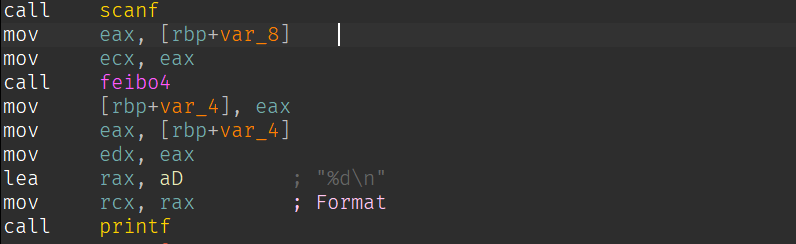
调试器版本：GNU gdb (GDB) 11.2

1. ❯ gcc -c main.c -o main.obj
2. ❯ gcc -c feibo4.c -o feibo4.obj
3. ❯ gcc main.obj feibo4.obj -o a.exe

运行后测试输入，当n > 2时，程序无输出，使用GDB调试工具调试运行，当输入数据大于2时，GDB工具报错显示寄存器错误.

1. [Thread 12892.0xb2c exited with code 3221225725]
3. Program terminated with signal SIGSEGV, Segmentation fault.
4. The program no longer exists.

使用IDA工具打开main.obj文件



分析后发现，在windows环境下feibo4函数的入口参数使用ecx寄存器传递，而非edi寄存器，出口参数仍使用eax寄存器传递。因此需要修改内联汇编部分的入口参数传递方式，才能使其在windows环境中正常运行。

修改后的源文件feibo4\_windows.c如下：

1. // 在windows环境中入口参数使用ecx寄存器传递，与unix系统存在差异
2. **int** feibo4(**int** n)           // 入口参数ecx，出口参数eax
3. {
4. **int** result;
5. asm **volatile**(           // 内联汇编部分，入口参数n使用内存指针n传递，出口参数使用内存指针result传递
6. "mov %1, %%eax;"    // 从指向n的内存地址取出n的值传入eax中
7. "cmp $2, %%eax;"    // 比较n与2的大小
8. "jg r;"             // n大于2时递归计算
9. "mov $1, %%eax;"    // n小于等于2时直接返回1
10. "jmp endf;"
11. "r:;"
12. "sub $1, %%eax;"    // n - 1
13. "mov %%eax, %%ecx;" // 将n - 1传递给ecx寄存器，函数入口参数
14. "call feibo4;"      // feibo4(n-1)
15. "mov %%eax, %0;"    // 将函数出口参数保存到result所在的内存中
16. "mov %1, %%eax;"    // 从指向n的内存地址取出n的值传入eax中
17. "sub $2, %%eax;"    // n - 2
18. "mov %%eax, %%ecx;" // 将n - 2传递给ecx寄存器，函数入口参数
19. "call feibo4;"      // feibo4(n-2)
20. "add %0, %%eax;"    // %0中储存的为feibo4(n-1)的值，将其加到feibo4(n-2)的计算结果中
21. // 即eax中储存的为feibo4(n-1)+feibo4(n-2)
22. "endf:"
23. "mov %%eax, %0;"    // 将返回值传送到result所在的内存中
24. : "=m"(result)      // OUTPUT出口参数，使用内存传参，通过%0调用
25. : "m"(n));          // INPUT入口参数，使用内存传参，通过%1调用
26. **return** result;
27. }

编译运行后输入测试数据：

1. PS C:\MyPC\tmp\masm\gcc\_inline\_asm> gcc main.c .\feibo4\_windows.c -o a.exe
2. PS C:\MyPC\tmp\masm\gcc\_inline\_asm> ./a.exe
3. 8
4. 21
5. PS C:\MyPC\tmp\masm\gcc\_inline\_asm> ./a.exe
6. 11
7. 89

n = 8时，输出结果21，n = 11时，输出结果89，能够得到正常计算结果。

1. 总结与体会

在实验的过程中，学习了汇编语言程序的编写方法，通过在debug工具下调试汇编程序，加深了对汇编程序运行过程的了解，基本掌握了常用汇编指令的使用。相较于常见的高级语言如c、python，汇编语言编写效率明显较低，并且汇编语言的可读性更差，平台间移植困难，虽然有着这么多的缺点，但是汇编其直接操作底层硬件的有点有着很高的灵活性，可以通过不同的命令组合实现相同的功能，也可以使用较少的代码来直接操作硬件，从而节约存储空间等资源开销。此外，汇编还有着运行效率高的优势，在优化高级语言编写的程序时，可以将关键部分代码用汇编语言编写，从而提高程序的执行效率和减少资源开销。

虽然汇编语言已经不适合在实际开发环境中使用，但学习汇编语言仍然是有必要的，对于网安专业，在逆向分析中，经过混淆编译的程序难以得到源码，通过反汇编将其转换为汇编代码，从汇编代码中分析其运行逻辑，在这个过程中，需要对汇编语言有着较高的掌握程度，要了解各条命令的作用、函数调用方式等等。

汇编语言语言的学习，我认为更多地是为其他课程打基础，在今后的课程中，我们还将学习操作系统、编译原理等课程，汇编语言程序设计这门课可以帮助我们更深入地学习其他课程。