合肥工业大学

计算机与信息学院

实验报告

课程：汇编语言程序设计

专业班级：计科21-4班

学号：2021218189

姓名：邢智博

实验一

第一题

> 1. 利用DEBUG程序中的“E”命令，将两个多字节数“12345678H”和“FEDCBA98H”分别送入起始地址为DS:0200H和DS:0204H两个单元中；实现将DS:0200H单元和DS:0204H单元中的数据相加，并将运算结果存放在DS:0208H单元中。

首先在debug之中使用e命令输入如下的数据：

```
C:\>debug
-e ds:0200
073F:0200  00.78   00.56   00.34   00.12   00.98   00.ba   00.dc   00.fe

-d ds:0200
073F:0200  78 56 34 12 98 BA DC FE-00 00 00 00 00 00 00 00   xV4.............
073F:0210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0250  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0260  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0270  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-S
```

之后，使用a命令输入如下的代码并且执行：

```
-a
073F:0100 mov bx,0200
073F:0103 mov ax,[bx]
073F:0105 add ax,[bx+4]
073F:0108 mov dx,[bx+2]
073F:010B adc dx,[bx+6]
073F:010E mov [bx+8],ax
073F:0111 mov [bx+a],dx
073F:0114
-g cs:0111

AX=1110  BX=0200  CX=0000  DX=1111  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0111   NV UP EI PL NZ AC PE CY
073F:0111 89570A        MOV     [BX+0A],DX                     DS:020A=0000
-S
```

查看内存之中的内容:

```
-d ds:0200
073F:0200  78 56 34 12 98 BA DC FE-10 11 11 11 00 00 00 00   xV4.............
073F:0210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0240  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0250  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0260  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
073F:0270  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

第二题

> 2. X、Y、Z、V均为字变量，在X、Y、Z、V字单元中存放是16位带符号数。试编写汇编语言程序完成以下功能：计算表达式值（V-（X*Y+Z-720））/X，将运算结果整数放在SUM1单元，余数放在SUM2单元。
>
> 实验提示（参考第3章和Debug部分的PPT）：
>
> （1） 在DOSBOX编译链接成可执行文件后，使用Debug装入内存。
>
> （2） 使用U命令反汇编代码，并与源文件比对，需要注意数据段名、变量转入内存后的形式。
>
> （3） 分别在G命令执行前后，使用D命令查看各个变量的值。

1. 汇编源程序

```
assume cs:code,ds:data
data segment
    x dw 12
    y dw -123
    z dw 1234
    v dw 4312
    sum1 dw 0
    sum2  dw 0
data ends

code segment
```

```
    mov ax,data
    mov ds,ax

    mov ax,x
    imul y    ; 两个十六位数相乘会将计算结果的高位放在dx之中，低位放在ax之中
    add ax,z
    adc dx,0
    sub ax,720
    sbb dx,0  ; 减去借位
    mov cx,0
    mov bx,v
    sub bx,ax
    sbb cx,0
    mov dx,cx
    mov ax,bx
    idiv word ptr x
    mov sum1,ax
    mov sum2,dx
code ends
end
```

2. 编译链接，在debug之中进行调试：

```
C:\>debug exper2.exe
-u
076A:0000 0C00          OR      AL,00
076A:0002 85FF          TEST    DI,DI
076A:0004 D204          ROL     BYTE PTR [SI],CL
076A:0006 D810                  FCOM    DWORD PTR [BX+SI]
076A:0008 0000          ADD     [BX+SI],AL
076A:000A 0000          ADD     [BX+SI],AL
076A:000C 0000          ADD     [BX+SI],AL
076A:000E 0000          ADD     [BX+SI],AL
076A:0010 B86A07        MOV     AX,076A
076A:0013 8ED8          MOV     DS,AX
076A:0015 A10000        MOV     AX,[0000]
076A:0018 F72E0200      IMUL    WORD PTR [0002]
076A:001C 03060400      ADD     AX,[0004]
```

```
076A:001C 03060400      ADD     AX,[0004]
-u
076A:0020 83D200        ADC     DX,+00
076A:0023 2DD002        SUB     AX,02D0
076A:0026 83DA00        SBB     DX,+00
076A:0029 B90000        MOV     CX,0000
076A:002C 8B1E0600      MOV     BX,[0006]
076A:0030 2BD8          SUB     BX,AX
076A:0032 83D900        SBB     CX,+00
076A:0035 8BD1          MOV     DX,CX
076A:0037 8BC3          MOV     AX,BX
076A:0039 F73E0000      IDIV    WORD PTR [0000]
076A:003D A30800        MOV     [0008],AX
-S_
```

3. 执行

```
-g cs:0044

AX=EC63  BX=149A  CX=FFFF  DX=FFF6  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076A  IP=0044   NV UP EI NG NZ AC PE CY
076A:0044 0000          ADD     [BX+SI],AL                      DS:149A=00
-S
```

第三题

> BL中的只有一位为0，编写程序测试0所在的位数（从左编号，最左边为第0位），并输出提示信息“The X Bit is 0”（X为0、1、2、3...7），要求使用地址表方法实现。

1. 汇编源程序

```
stack segment stack
    dw 512 dup(0)
stack ends
data segment
    space dw code0,code1,code2,code3,code4,code5,code6,code7,code8    ; 偏移地址表
    zero db 'The x bit is 0 $'  ;应该还需要添加结束字符$ or 0
    one db 'the x bit is 1 $'
    two db 'the x bit is 2 $'
    three db 'the x bit is 3 $'
    four db 'the x bit is 4 $'
    five db 'the x bit is 5 $'
    six db 'the x bit is 6 $'
    seven db 'the x bit is 7 $'
    eight db 'there have no zero $'
data ends

code segment
assume cs:code,ss:stack,ds:data
start:
    ; 感觉还是很机械的写法，一条一条的罗列
    mov ax,data
    mov ds,ax
    mov bl,11110111B
    mov si,0
    mov cx,8
    mov al,00000001B
again:
    test bl,al
    jz foundzero
    inc si
    inc si
    shl al,1
    loop again
foundzero:
    jmp space[si]
```

```
36
37 code0:
38     mov dx,offset zero
39     jmp print
40 code1:
41     mov dx,offset one
42     jmp print
43 code2:
44     mov dx,offset two
45     jmp print
46 code3:
47     mov dx,offset three
48     jmp print
49 code4:
50     mov dx,offset four
51     jmp print
52 code5:
53     mov dx,offset five
54     jmp print
55 code6:
56     mov dx,offset six
57     jmp print
58 code7:
59     mov dx,offset seven
60     jmp print
61 code8:
62     mov dx,offset eight
63     jmp print
64 print:
65     ; 调用int21h的9号功能
66     mov ah,09h
67     int 21h
68     mov ax,4c00h
69     int 21h
70
71 finish:
72     jmp space[si]
73
74 code ends
75 end start
```

2. 编译链接，在debug之中查看数据在debug之中的存储形式，如下图：

```
-d
075A:0100  10 00 21 00 32 00 43 00-54 00 65 00 76 00 87 00   ..!.2.C.T.e.v...
075A:0110  54 68 65 20 78 20 62 69-74 20 69 73 20 30 0D 0A   The x bit is 0..
075A:0120  24 74 68 65 20 78 20 62-69 74 20 69 73 20 31 0D   $the x bit is 1.
075A:0130  0A 24 74 68 65 20 78 20-62 69 74 20 69 73 20 32   .$the x bit is 2
075A:0140  0D 0A 24 74 68 65 20 78-20 62 69 74 20 69 73 20   ..$the x bit is
075A:0150  33 0D 0A 24 74 68 65 20-78 20 62 69 74 20 69 73   3..$the x bit is
075A:0160  20 34 0D 0A 24 74 68 65-20 78 20 62 69 74 20 69    4..$the x bit i
075A:0170  73 20 35 0D 0A 24 74 68-65 20 78 20 62 69 74 20   s 5..$the x bit
-r
AX=FFFF  BX=0000  CX=00A9  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076A  IP=0000   NV UP EI PL NZ NA PO NC
076A:0000 1000          ADC     [BX+SI],AL                      DS:0000=CD
```

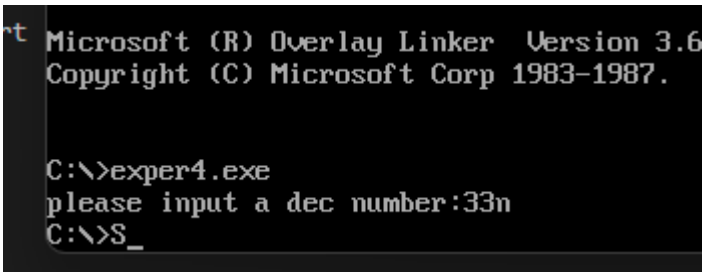
可以发现：第一行存储了每个数组成员相对起始地址的相对偏移量，所以可以使用`ds:[bx+idata]`方式访问表中的数据

3. 运行程序，查看运行结果

```
Z:\>mount c d:debug
Drive C is mounted as local directory d:debug\

Z:\>c:

C:\>masm exper3;
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.


  51630 + 464914 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors

C:\>link exper3;

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


C:\>exper3.exe
the x bit is 3
C:\>S
```

第四题

> 在内存Score缓冲区中存放有100个学生的成绩数据，为无符号字节数。设 计程序完成如下功能：根据用户输入的一个2位十进制数，作为查找对象，在该数组中查找，若找到则显示“Y”，若没找到则显示“N”。



运行结果：输入33，输出n

代码：

```
data segment
    score    DB 100 dup(?)
    MsgY     DB 'y$'
    MsgN     DB 'n$'
    MsgInput db 'please input a dec number:$'
data ends
stack segment stack
          db 510 dup(0)
stack ends
code segment
             assume cs:code,ds:data,ss:stack
    start:
             mov    ax,data
             mov    ds,ax
             mov    ax,stack
             mov    ss,ax

             lea    dx,MsgInput
             mov    ah,09h
             int    21h

    ; 调用int21h中断的01号功能获取用户输入字符,输入字符存入到al之中
             mov    ah,01h
             int    21h

             mov    dl,al
             mov    ah,01h
             int    21h
             mov    bl,al
             mov    ax,0
             mov    al,dl
             mov    dl,0ah
             mul    dl
             add    al,bl

             mov    cl,100
             mov    bx,offset score
             mov    si,0
    search:
             mov    dl,[bx+si]
```

```
            inc     si
            cmp     al,dl
            je      printYes
            LOOP    search

    ; 循环结束以后还是没有找到，那么输出MsgN
            lea     dx,MsgN
            mov     ah,09h
            int     21h

            mov     ax,4c00h
            int     21h



    printYes:
            lea     dx,MsgY
            mov     ah,09h
            int     21h

            mov     ax,4c00h
            int     21h




            mov     ax,4c00h
            int     21h
code ends
end start
```

实验二

第一题

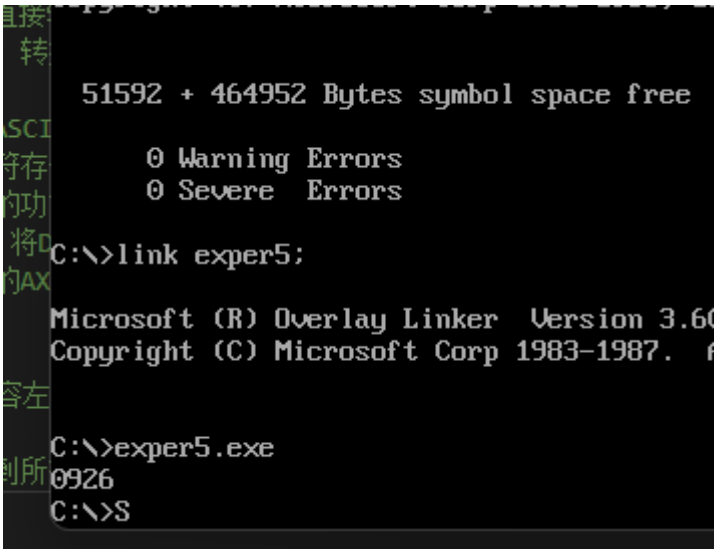
> 1、编写一个子程序计算z=f(x,y)=x*y -2x+2022（x,y,z有符号数字变量），要求：
>
> （1）假设x*y的结果还是16位；
>
> （2）输入参数通过堆栈传送，输出参数通过AX传送；
>
> （3）主程序调用后，结果需要输出到屏幕上。

```
;1、编写一个子程序计算z=f(x,y)=x*y -2x+2022（x,y,z有符号数字变量），要求：
; （1）假设x*y的结果还是16位；
; （2）输入参数通过堆栈传送，输出参数通过AX传送；
```

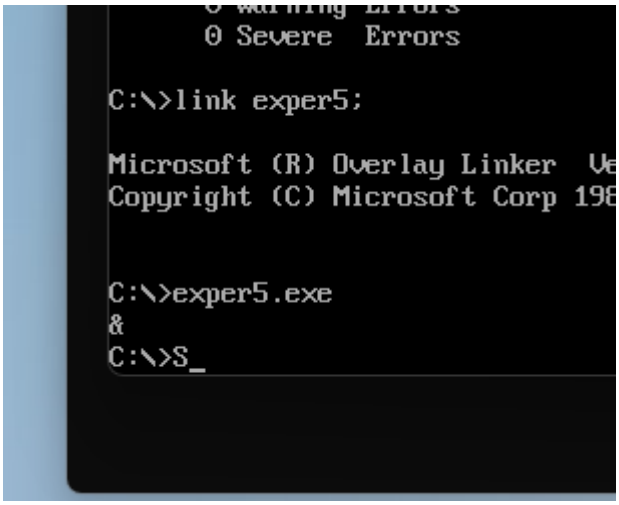
```
; （3）主程序调用后，结果需要输出到屏幕上。

stack segment stack
          dw 512 dup(0)
stack ends
data segment
         dw 16 dup(0)
data ends
code segment
                  assume ss:stack,cs:code,ds:data
    start:
                  mov    ax,data
                  mov    ds,ax
                  mov    ax,stack
                  mov    ss,ax
                  mov    bx,12
                  mov    sp,128h
                  push   bx
                  mov    ax,32
                  push   ax
                  mov    bx,0
                  call   func

                  MOV    CX, 4                          ; 设置循环计数器，因为一个十六进制数需要输出4个字符



    PrintHexDigit:
                  push   ax
                  and    ah,0f0h
                  push cx
                  mov cl,4
                  shr ah,cl
                  pop cx
                  cmp    ah,9
                  jbe    PrintDigit                     ; 如果是数字，直接输出
                  add    ah, 7                          ; 如果不是数字，转换成A~F字符
    PrintDigit:
                  add    ah, 30H                        ; 将数字转换成ASCII码
                  mov    dl, ah                         ; 将要输出的字符存储在DL寄存器中
                  mov    ah,02h                         ; 设置输出字符的功能号
                  int    21h                            ; 调用21H中断，将DL寄存器中的值输出到屏幕上
                  pop    ax                             ; 弹出栈中保存的AX寄存器中的值
                  push   CX
                  mov    cl,4
                  shl    ax,cl                          ; 将ax之中的内容左移四位
                  pop    cx
                  LOOP   PrintHexDigit                  ; 循环执行，直到所有字符输出完成

```

```
                    mov     ax,4c00h
                    int     21h


    func:           push    bp
                    mov     bp,sp
                    sub     sp,64h
                    mov     ax,[bp+4]
                    mov     bx,[bp+6]                   ; 取出两个数值。
                    imul    bx
                    mov     dx,0                        ; 因为假设了结果依旧是十六位数字，所以无需使用dx
                    mov     dx,ax
                    mov     ax,[bp+4]
                    add     ax,ax
                    neg     ax
                    add     dx,ax
                    add     dx,2022
                    mov     ax,dx

                    mov     bx,0                        ; 恢复现场，将bx置0

                    add     sp,64h
                    pop     bp
                    ret
code ends
end start
```

输出结果：

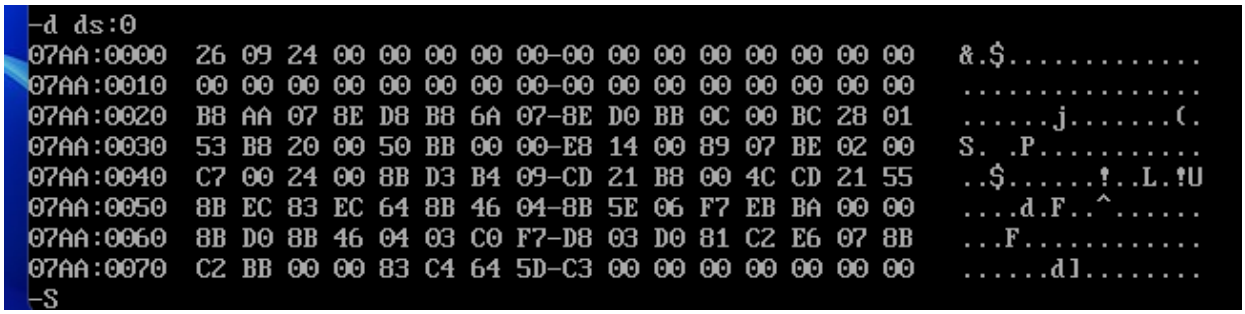


实验过程：



第一次进行调试输出的结果是'&'，怀疑有两种可能：

- 将计算好的结果作为ASCII码进行了输出。但是按理说计算结果会是一个大于2022的数字，不会解析为一个ASCII字符
- 在子程序之中读取堆栈数据的位置不对



使用debug调试程序验证了猜想：输出过程中将数字解释为了字符串，所以会按ASCII进行解释。

下面编写输出数字程序。

我最初的设想是通过堆栈返回多个参数，但是由于在调用子函数以及从子函数返回的过程之中都需要`push ip`与`pop ip`，这意味着通过堆栈传递多个返回值是错误的，这也对应着c/c++要求函数返回值只能是一个的原因。

输出单个字符可以调用`int 21h`终端的02号指令，现在我们需要做的就是将数字的每一位转换成为对应的ASCii码（在原来的数字上+30h），之后送入到02号终端程序之中。这一部分的代码如下：

```
PrintHexDigit:
                push    ax
                and     ah,0f0h
                push cx
                mov cl,4
                shr ah,cl
                pop cx
                cmp     ah,9
                jbe     PrintDigit                  ; 如果是数字，直接输出
                add     ah, 7                       ; 如果不是数字，转换成A~F字符
   PrintDigit:
                add     ah, 30H                     ; 将数字转换成ASCII码
```

```
                    mov     dl, ah                          ; 将要输出的字符存储在DL寄存器中
                    mov     ah,02h                          ; 设置输出字符的功能号
                    int     21h                             ; 调用21H中断，将DL寄存器中的值输出到屏幕上
                    pop     ax                              ; 弹出栈中保存的AX寄存器中的值
                    push    cx
                    mov     cl,4
                    shl     ax,cl                           ; 将ax之中的内容左移四位
                    pop     cx
                    LOOP    PrintHexDigit                   ; 循环执行，直到所有字符输出完成

                    mov     ax,4c00h
                    int     21h
```

大作业

一、分析 C 语言程序对应的反汇编代码，要求：

1）除 Main 函数外，至少还包含一个含有两个以上输入参数的函数，参数和返回值均为整形数，函数中需要包含各算术运算和 if（或 while）语句。

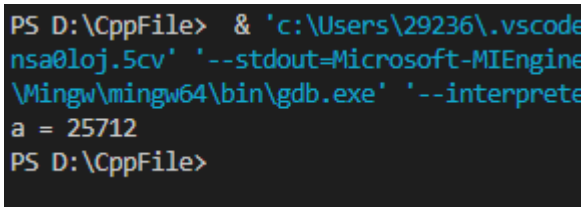
2）嵌入一段汇编代码，至少包含两个位运算（逻辑运算或移位运算），位运算中的变量使用 C 语言定义。（选做）

C源程序如下：

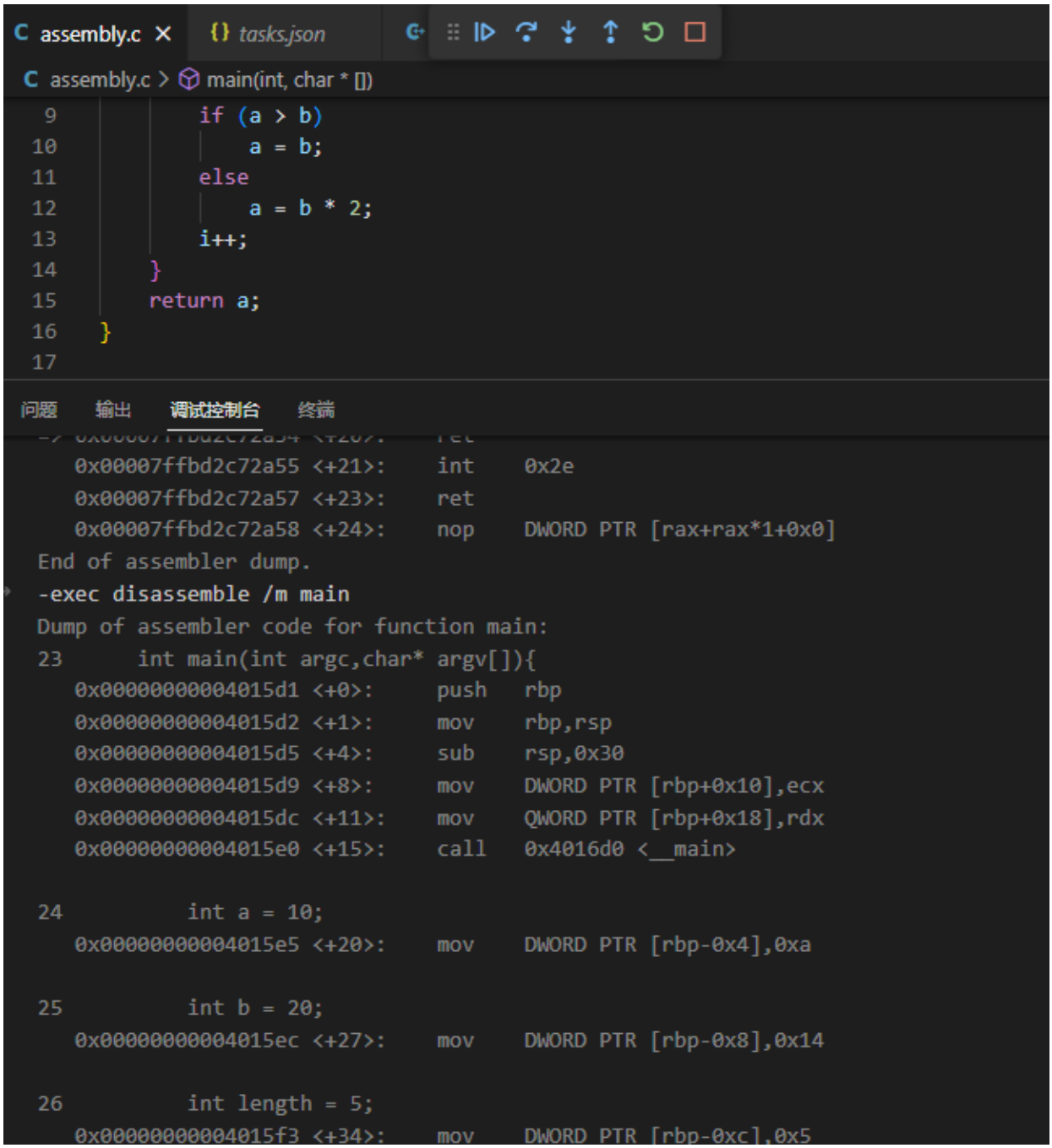
```c
#include <stdio.h>

int func(int a, int b, int length)
{
    int i = 0;
    while (i < length)
    {
        a++;
        b = a * 2 - a / 4 + i + 1;
        b--;
        if (a > b)
            a = b;
        else
            a = b * 2;
        i++;
    }
    return a;
}

int main(int argc, char *argv[])
{
```

```
    int a = 10;
    int b = 20;
    int length = 5;
    a = func(a, b, length);
    __asm__ __volatile__(
        "shl %%eax \n\t"
        "addl %%ebx,%%eax \n\t"
        "and %%eax,%%ebx \n\t"
        "shl %%eax"
        :"=a"(a)
        :"a"(a),"b"(b));
    printf("a = %d",a);
}
```

运行结果:



在VS code之中进行反汇编，查看汇编代码如下：



`main()`函数的汇编代码:

```
-exec disassemble /m main
Dump of assembler code for function main:
24  {
   0x00000000004015e3 <+0>: push   rbp
   0x00000000004015e4 <+1>: push   rbx                                ; 保存现场
   0x00000000004015e5 <+2>: sub    rsp,0x38                           ; 开辟栈空间
   0x00000000004015e9 <+6>: lea    rbp,[rsp+0x80]                     ; 将rsp+80h的位置付给rbp
   0x00000000004015f1 <+14>:    mov    DWORD PTR [rbp-0x30],ecx
   0x00000000004015f4 <+17>:    mov    QWORD PTR [rbp-0x28],rdx       ; 将参数保存到堆栈
```

```
   0x00000000004015f8 <+21>:    call   0x401710 <__main>

25      int a = 10;
=> 0x00000000004015fd <+26>:    mov    DWORD PTR [rbp-0x54],0xa             ; 声明变量

26      int b = 20;
   0x0000000000401604 <+33>:    mov    DWORD PTR [rbp-0x58],0x14

27      int length = 5;
   0x000000000040160b <+40>:    mov    DWORD PTR [rbp-0x5c],0x5

28      a = func(a, b, length);
   0x0000000000401612 <+47>:    mov    ecx,DWORD PTR [rbp-0x5c]
   0x0000000000401615 <+50>:    mov    edx,DWORD PTR [rbp-0x58]
   0x0000000000401618 <+53>:    mov    eax,DWORD PTR [rbp-0x54]             ; 通过寄存器传递参数
   0x000000000040161b <+56>:    mov    r8d,ecx
   0x000000000040161e <+59>:    mov    ecx,eax                              ; 可能是eax需要保存返回值以及在子函数之中需要使用,
                                                                            ; 提前将eax之中的内容保存到ecx
   0x0000000000401620 <+61>:    call   0x401550 <func>                      ; 调用函数
   0x0000000000401625 <+66>:    mov    DWORD PTR [rbp-0x54],eax

29      __asm__ __volatile__(
   0x0000000000401628 <+69>:    mov    eax,DWORD PTR [rbp-0x54]
   0x000000000040162b <+72>:    mov    edx,DWORD PTR [rbp-0x58]             ; 将a和b保存到eax与ebx之中,以方便内联汇编之中使用
   0x000000000040162e <+75>:    mov    ebx,edx                              ; 将b移动到ebx
   0x0000000000401630 <+77>:    shl    eax,1
   0x0000000000401632 <+79>:    add    eax,ebx
   0x0000000000401634 <+81>:    and    ebx,eax
   0x0000000000401636 <+83>:    shl    eax,1                                ; 以上四行是嵌入的汇编代码
   0x0000000000401638 <+85>:    mov    DWORD PTR [rbp-0x54],eax             ; 保存a,对应'=a'部分

30          "shl %%eax \n\t"
31          "addl %%ebx,%%eax \n\t"
32          "and %%eax,%%ebx \n\t"
33          "shl %%eax"
34          :"=a"(a)
35          :"a"(a),"b"(b));
36      printf("a = %d",a);
   0x000000000040163b <+88>:    mov    eax,DWORD PTR [rbp-0x54]             ; 将a保存到eax之中
   0x000000000040163e <+91>:    mov    edx,eax
   0x0000000000401640 <+93>:    lea    rcx,[rip+0x29b9]        # 0x404000   ; 理论上来讲,应该是取输出字符串的偏移地址.之后传给printf
```

```
   0x0000000000401647 <+100>:   call   0x402b40 <printf>                    ; 调用printf来进行输出
   0x000000000040164c <+105>:   mov    eax,0x0

37  }
   0x0000000000401651 <+110>:   add    rsp,0x38
   0x0000000000401655 <+114>:   pop    rbx
   0x0000000000401656 <+115>:   pop    rbp                                  ; 函数结束,还原现场
   0x0000000000401657 <+116>:   ret

End of assembler dump.
```

这一部分为func函数的汇编代码：

```
Dump of assembler code for function func:
4   {
   0x0000000000401550 <+0>: push   rbp                                      ; 将rbp压栈，保存现场
   0x0000000000401551 <+1>: mov    rbp,rsp                                  ;
   0x0000000000401554 <+4>: sub    rsp,0x10                                 ; 栈顶指针上移,开辟栈空间
   0x0000000000401558 <+8>: mov    DWORD PTR [rbp+0x10],ecx                 ; ecx,edx与r8d致中保存了传进来的参数.将参数保存到栈空间
   0x000000000040155b <+11>:    mov    DWORD PTR [rbp+0x18],edx
   0x000000000040155e <+14>:    mov    DWORD PTR [rbp+0x20],r8d             ; r8是x64架构的64位寄存器,r8d是其低三十二位字节

5       int i = 0;
   0x0000000000401562 <+18>:    mov    DWORD PTR [rbp-0x4],0x0              ; 声明变量

6       while (i < length)                                                  ; 跳到比较大小部分
   0x0000000000401569 <+25>:    jmp    0x4015d2 <func+130>
   0x00000000004015d2 <+130>:   mov    eax,DWORD PTR [rbp-0x4]              ; 注意这两行的ip为130和133.也就是说,如果不满足while的条
                                                                           ; 件,会跳过循环体,直接向后边执行.这样设计的另一个好处是,
                                                                           ; 执行完循环部分内容以后,可以自动将ip指向条件判断部分
   0x00000000004015d5 <+133>:   cmp    eax,DWORD PTR [rbp+0x20]             ; 判断i是否小于length
   0x00000000004015d8 <+136>:   jl     0x40156b <func+27>                   ; 如果小于,接着向下执行

7       {
8           a++;
   0x000000000040156b <+27>:    add    DWORD PTR [rbp+0x10],0x1             ; 自增
```

```
24
25  9           b = a * 2 - a / 4 + i + 1;
26     0x000000000040156f <+31>:    mov    eax,DWORD PTR [rbp+0x10]          ;
   将参数a保存到寄存器之中
27     0x0000000000401572 <+34>:    lea    ecx,[rax+rax*1]                   ;
   实现a*2.lea为load effictive address的含义,会取得
28                                                                            ;
   rax之中的内容.
29     0x0000000000401575 <+37>:    mov    eax,DWORD PTR [rbp+0x10]          ;
   将a保存到寄存器之中
30     0x0000000000401578 <+40>:    lea    edx,[rax+0x3]                     ;
   从rax的第三个字节开始取内容,相当于a左移两位,也就是除以4
31     0x000000000040157b <+43>:    test   eax,eax                           ;
   判断正负
32     0x000000000040157d <+45>:    cmovs  eax,edx                           ;
   如果eax最高位为1,也就是负数,将其移动到edx之中
33     0x0000000000401580 <+48>:    sar    eax,0x2                           ;
   这几步实现了a/4
34     0x0000000000401583 <+51>:    neg    eax                               ;
   取反
35     0x0000000000401585 <+53>:    lea    edx,[rcx+rax*1]                   ;
   将a*2-a/4保存到edx之中
36     0x0000000000401588 <+56>:    mov    eax,DWORD PTR [rbp-0x4]           ;
   将i保存到eax
37     0x000000000040158b <+59>:    add    eax,edx                           ;
   加上i
38     0x000000000040158d <+61>:    add    eax,0x1                           ;
   加上立即数,1
39     0x0000000000401590 <+64>:    mov    DWORD PTR [rbp+0x18],eax
40
41  10          b--;
42     0x0000000000401593 <+67>:    sub    DWORD PTR [rbp+0x18],0x1
43
44  11          b-=1;
45     0x0000000000401597 <+71>:    sub    DWORD PTR [rbp+0x18],0x1          ;
   自减.这两种写法在底层是同样的.
46
47  12          a = a+i%2;
48     0x000000000040159b <+75>:    mov    edx,DWORD PTR [rbp-0x4]           ;
   将i保存到edx之中
49     0x000000000040159e <+78>:    mov    eax,edx                           ;
   将i保存到eax之中
50     0x00000000004015a0 <+80>:    sar    eax,0x1f                          ;
   算数右移31位,取得符号位
51     0x00000000004015a3 <+83>:    shr    eax,0x1f                          ;
   先算数右移,再逻辑右移
52     0x00000000004015a6 <+86>:    add    edx,eax                           ;
   将
53     0x00000000004015a8 <+88>:    and    edx,0x1                           ;
54     0x00000000004015ab <+91>:    sub    edx,eax
55     0x00000000004015ad <+93>:    mov    eax,edx
56     0x00000000004015af <+95>:    add    DWORD PTR [rbp+0x10],eax
```

```
13          a = a^4;
   0x00000000004015b2 <+98>:    xor    DWORD PTR [rbp+0x10],0x4  `          ;
与4进行异或

14          if (a > b)
   0x00000000004015b6 <+102>:   mov    eax,DWORD PTR [rbp+0x10]             ;
取得a与b
   0x00000000004015b9 <+105>:   cmp    eax,DWORD PTR [rbp+0x18]
   0x00000000004015bc <+108>:   jle    0x4015c6 <func+118>                  ;
如果不满足判断条件,那么跳转到else部分

15              a = b;
   0x00000000004015be <+110>:   mov    eax,DWORD PTR [rbp+0x18]
   0x00000000004015c1 <+113>:   mov    DWORD PTR [rbp+0x10],eax             ;
将b的数值赋给a
   0x00000000004015c4 <+116>:   jmp    0x4015ce <func+126>                  ;
跳过else

16          else
17              a = b * 2;
   0x00000000004015c6 <+118>:   mov    eax,DWORD PTR [rbp+0x18]             ;
取得b
   0x00000000004015c9 <+121>:   add    eax,eax                              ;
*2
   0x00000000004015cb <+123>:   mov    DWORD PTR [rbp+0x10],eax             ;
将b的数值赋给a

18          i++;
   0x00000000004015ce <+126>:   add    DWORD PTR [rbp-0x4],0x1              ;
自增

19      }
20      return a;
   0x00000000004015da <+138>:   mov    eax,DWORD PTR [rbp+0x10]             ;
将返回值保存到eax之中.汇编通过eax来返回参数

21  }
   0x00000000004015dd <+141>:   add    rsp,0x10
   0x00000000004015e1 <+145>:   pop    rbp                                  ;
还原现场
   0x00000000004015e2 <+146>:   ret

End of assembler dump.
```

注意到在`main()`函数刚刚开始的执行的时候调用的`__main`函数,查看`__main`函数的汇编代码如下:

```
Dump of assembler code for function __main:
   0x00000000004016d0 <+0>: mov    eax,DWORD PTR [rip+0x595a]        # 0x407030
<initialized>
   0x00000000004016d6 <+6>: test   eax,eax
   0x00000000004016d8 <+8>: je     0x4016e0 <__main+16>
   0x00000000004016da <+10>:    ret
   0x00000000004016db <+11>:    nop    DWORD PTR [rax+rax*1+0x0]
   0x00000000004016e0 <+16>:    mov    DWORD PTR [rip+0x5946],0x1        #
0x407030 <initialized>
   0x00000000004016ea <+26>:    jmp    0x401660 <__do_global_ctors>
   0x00000000004016ef <+31>:    nop
End of assembler dump.
```