OSLab01：大数相加

完成记录

时间	完成项目	后期计划
2021-10-6	完成输入输出函数，实现了对两个数的分隔	实现简单加法
2021-10-7	完成简单大数加法，同时实现了对寄存器中数字的打印函数 `printInt`	修改进位不正常的bug（考虑通过输出函数修改而不是改变逻辑）
2021-10-8	完成了大数加法	写出大数乘法的简单雏形
2021-10-13	完成了大数乘法	对负数的支持（即完成一个减法）
2021-10-14	完成了对负数的支持	

实验环境

我使用的

> 参考*win10搭建x86汇编编程环境（st3+nasm+bochs）*，有改动

- 使用VMware配置Ubuntu16.04虚拟机，通过共享文件夹实现在主机上编码
- 使用Sublime Text+插件实现代码高亮

重点安利

https://github.com/Dman95/SASM

SASM：一个支持大部分ASM汇编语言的IDE，有widows和Linux版。个人推荐使用Linux版，也支持图形界面。且能够自己完成输入输出函数。在Windows环境下输入输出函数不便于自己完成。

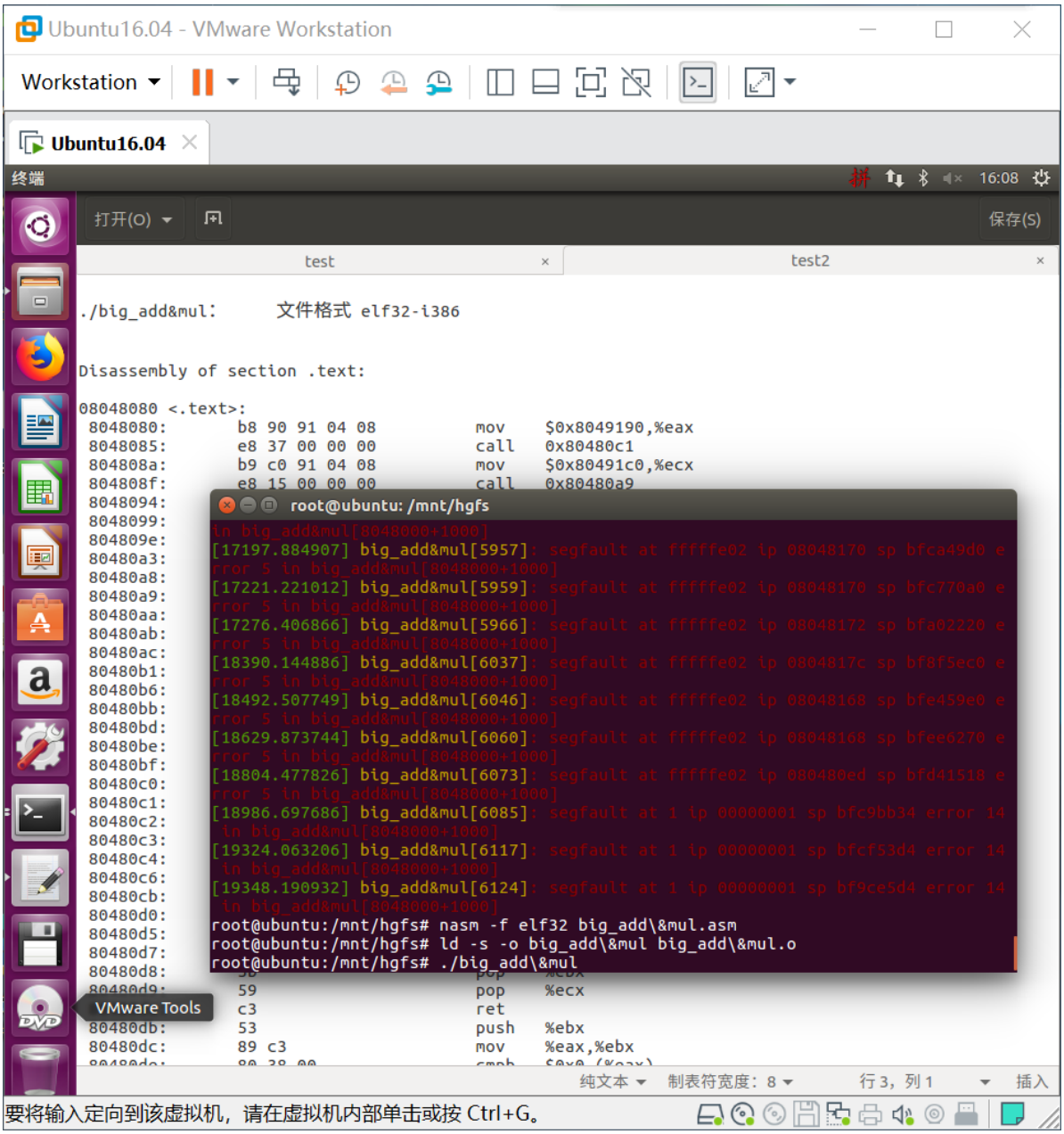
注意！SASM和手动编译结果可能存在不同，请务必进行终端测试（注：很有可能是下文"特定条件下的长度函数"的原因）

踩坑合集

段错误的调试方法

```
dmesg
objddump -d ./[] > [].test
```

两条指令，第一条是显示出段错误的保存信息，包括出错地址。第二条指令将代码编译成机器码，便于查找地址。



什么都没写报段错误？试试exit！

在你写的`_start`函数末尾加上

```
mov    ebx, 0
mov    eax, 1
int    80h
ret
;标准退出函数
```

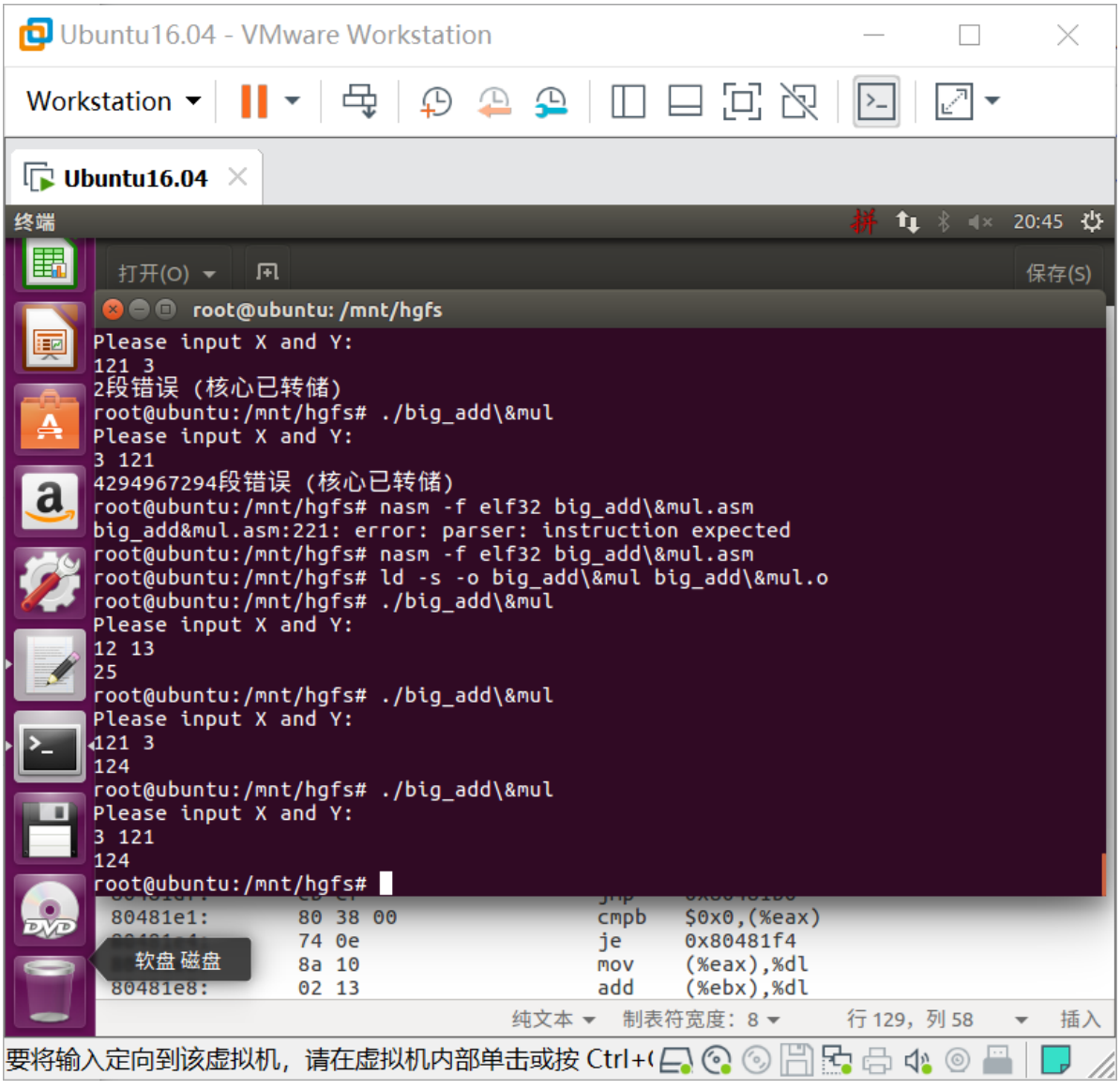
寄存器不存负数

看一段代码：

```
sub esi,edi;此时esi=num1_len,edi=num2_len
cmp esi,0
ja num1_longer
je ture_add
jmp num2_longer
```

看起来很正确？事实上esi和edi不相等时均会跳转到`num1_longer`。

事实上，我们打印出期望中应该是负数的esi的值，会发现它是4297964294



特定条件下的长度函数

经常使用到的长度函数是以0结尾的，但在我们这个特定的使用环境（即输入只有数字）的情况下，可能会有一些不同。建议出现问题时考虑将

```
cmp byte[eax],0
je xxx
```

改为

```
cmp byte[eax],'0'
jb xxx
cmp byte[eax],'9'
ja xxx
```

代码实现

输入输出

汇编语言的输入输出都是靠80h中断提供的。提供我的样例代码

输入

```
getInput:
;------------------------
;@用于接收用户输入,edx=缓冲区大小，ebx=想写入的文件
;@入口参数：ecx=变量的地址
;@出口参数：无
;------------------------
    push edx
    push ebx
    push eax
    mov edx,255
    mov ebx,0
    mov eax,3
    int 80h
    pop eax
    pop ebx
    pop edx
    ret
```

输出

```
printStr:
;------------------------
;@控制台输出一个字符串的函数
;@输入:eax=字符串的地址
;@输出:控制台输出
;------------------------
    push ecx
    push ebx
    push edx
    mov ecx,eax
    call strLen
```

```
    mov ebx,1;标准输出
    mov eax,4
    int 80h
    pop edx
    pop ebx
    pop ecx
    ret
```

加法

实现思路是先将长的数字长出来的部分移到结果上，再按位加，最后对结果进行规格化。

> 规格化的必要性：ascii码可存0~255，但只有48~57表示数字，故应该对其进行规格化，才能正确输出

加法

```
bigAdd:
;------------------------
;@将两个数字相加
;@输入num1，num2
;@输出改变result
;------------------------
    cmp byte[num_of_negative],1
    je bigSub
    push eax
    push ebx
    push edx
    mov ecx,result
    ;将result地址赋给ecx，可以通过修改ecx的值来改变result

    mov eax,num1
    mov ebx,num2
    mov esi,dword[num1_len]
    mov edi,dword[num2_len]
    cmp esi,edi
    ja num1_longer
    je ture_add
    jmp num2_longer
  num1_longer:
    sub esi,edi;此时esi保存长出来的长度
  num1_loop:
    cmp esi,0
    je ture_add
    mov dl,byte[eax]
    mov byte[ecx],dl
    inc eax
    inc ecx
    dec esi
    jmp num1_loop
  num2_longer:
    sub edi,esi;
  num2_loop:
    cmp edi,0
    je ture_add
    mov dl,byte[ebx]
    mov byte[ecx],dl
```

```
    inc ebx
    inc ecx
    dec edi
    jmp num2_loop
  ture_add:
    cmp byte[eax],0
    je finish_add
    mov dl,byte[eax]
    add dl,byte[ebx]
    sub dl,30h;减去ascii中对应的0
    mov byte[ecx],dl
    inc eax
    inc ebx
    inc ecx
    jmp ture_add
  finish_add:
    pop edx
    pop ebx
    pop eax
    ret
```

规格化

```
format_result:
;------------------------
;@将数字变成10进制的数字形式
;@输入eax=数字的地址
;@输出eax=改变后数字的地址，注如果发生进位，会提前输出一个1
;------------------------
        push edx
        mov ebx,eax
        ;dec ebx;数字从这里开始
    toEnd:
        cmp byte[eax],0
        je formatLoop
        inc eax
        jmp toEnd
    formatLoop:
        dec eax
        cmp eax,ebx
        je formatFinish
    formatLoop_1:
        cmp byte[eax],'9'
        jna formatLoop
        mov edx,eax
        dec edx
        sub byte[eax],10
        add byte[edx],1
        jmp formatLoop_1
    formatFinish:
        mov ecx,0
        cmp byte[eax],'9'
        ja extraPrint
        pop edx
        ret
```

```
    extraPrint:
        inc ecx
        sub byte[eax],10
        cmp byte[eax],'9'
        ja extraPrint
        push eax
        mov eax,ecx
        call printInt
        pop eax
        jmp formatFinish
```

乘法

乘法的思想是按位乘，按照以下公式：

设数字$a_ja_{j-1}a_{j-2}...a_0$、$b_ib_{i-1}b_{i-2}...b_0$,则有乘积

$$c_k = \sum_{m+n==k} a_mb_n \text{其中，} m \in j,\ n \in i$$

```
bigMul:
;------------------------
;@将两个数字相乘
;@输入num1，num2
;@输出改变multiple
;------------------------
    pushad
    mov ecx,multiple;通过更改ecx来更改multiple
    mov edx,num1
    mov ebx,num2
    mov esi,0
    mov edi,0
    ;移植需要实现将ecx放置在末尾
    mov eax,0
    add eax,dword[num1_len]
    add eax,dword[num2_len]
  init:
    cmp eax,0
    je toEnd_num1
    mov byte[ecx],'0'
    dec eax
    inc ecx
    jmp init
  toEnd_num1:
    cmp byte[edx+1],0
    je toEnd_num2
    inc edx
    jmp toEnd_num1
  toEnd_num2:
    cmp byte[ebx+2],0
    je multi
    inc ebx
    jmp toEnd_num2
  multi:
    push ebx
    ;初始时edx应该指向num1的末位
```

```
    ;test4:查验edx、ebx是否正确指向预计位置
    ;mov eax,0
    ;mov al,byte[ebx]
    ;cbw
    ;cwd
    ;call printInt
    ;finish test
    mov eax,0
    cmp edx,num1
    jb finish_multi
    mov al,byte[edx]

    sub al,30h
    jmp mul_loop
  mark:
    dec edx
    inc esi
    jmp multi
  mul_loop:

    cmp ebx,num2
    jb finish_mulLoop
    mov ah,byte[ebx]

    push edx
    mov dl,al
    sub ah,30h
    mul ah
    push ecx
    sub ecx,esi
    sub ecx,edi
    sub ecx,1
    cmp byte[ecx],150
    ja simple_format_byte
  finish_simple_format_byte:
    add byte[ecx],al
    pop ecx
    mov al,dl
    pop edx
    dec ebx
    inc edi
    jmp mul_loop
  finish_mulLoop:
    mov edi,0
    pop ebx
    jmp mark
  finish_multi:
    pop ebx
    popad
    ret
  simple_format_byte:
    sub byte[ecx],100
    add byte[ecx-2],1
    jmp finish_simple_format_byte
```

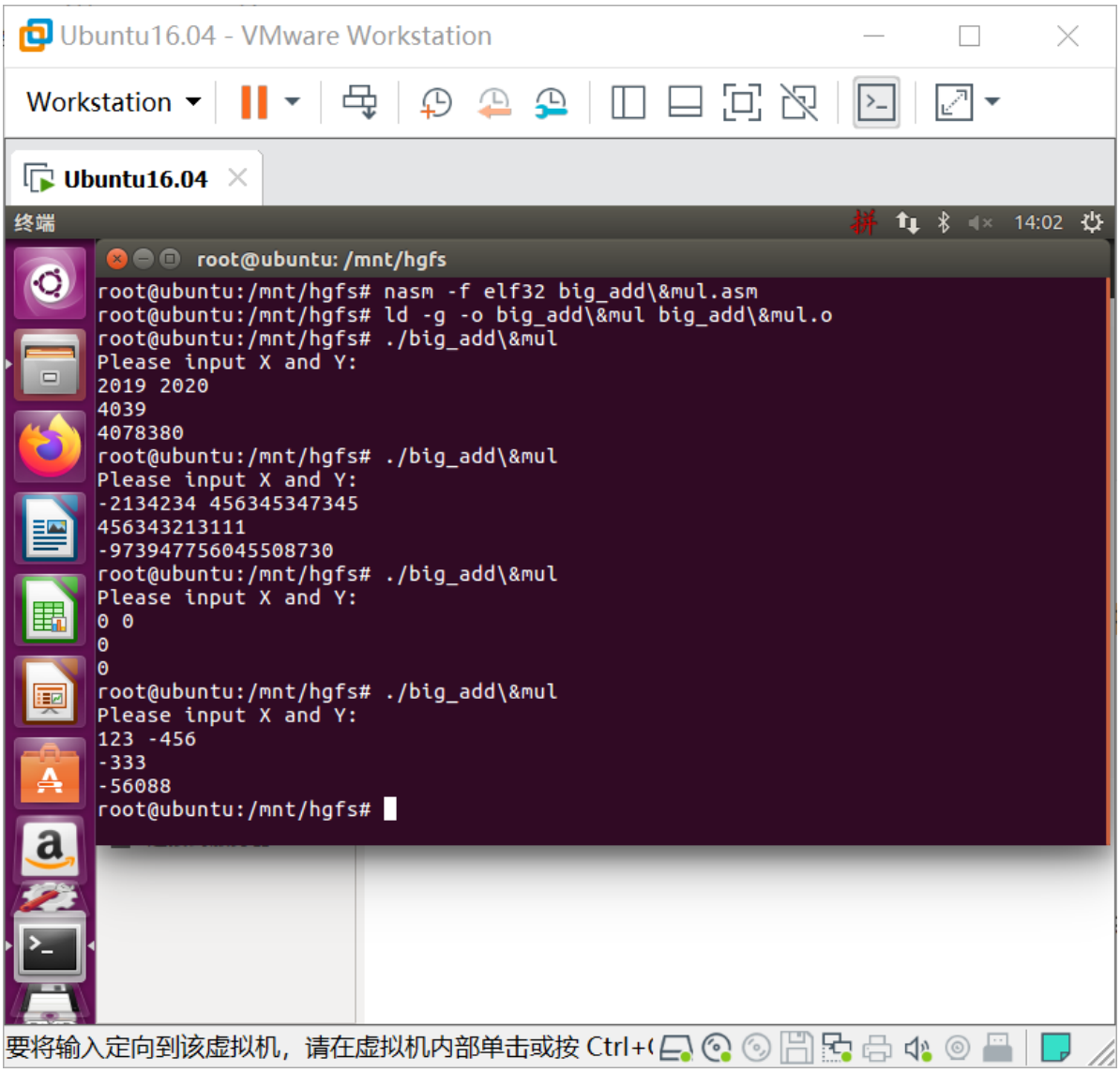
特别注意：简单规格化的意义

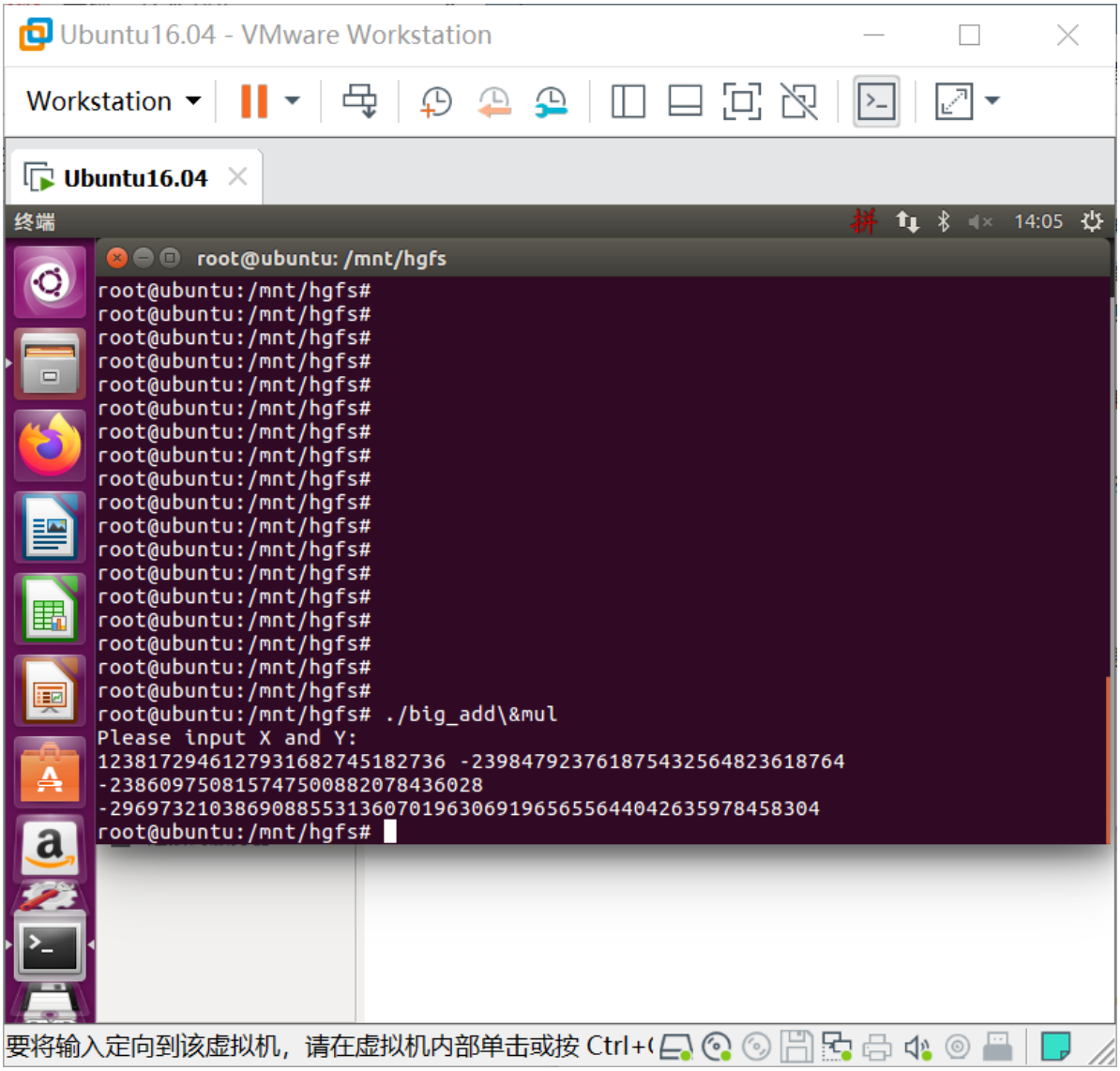
在加法中，9+9的ascii码并不会溢出，但是在乘法中是有溢出可能的。因此，我们需要在整体规格化之前先简单规格化一下，防止溢出。

负数实现

- 对乘法而言，只要记录现有负号的个数即可
- 对加法而言，单独实现一个减法，采用的方法和加法类似

运行截图




Ubuntu16.04 - VMware Workstation
Workstation
Ubuntu16.04
终端
14:05
root@ubuntu: /mnt/hgfs
root@ubuntu:/mnt/hgfs#
root@ubuntu:/mnt/hgfs# ./big_add\&mul
Please input X and Y:
123817294612793168274518273 6 -23984792376187543256482361876 4
-23860975081574750088207843602 8
-29697321038690885531360701963069196565564404263597845830 4
root@ubuntu:/mnt/hgfs#
要将输入定向到该虚拟机，请在虚拟机内部单击或按 Ctrl+(