1、实验任务一：Hello OS

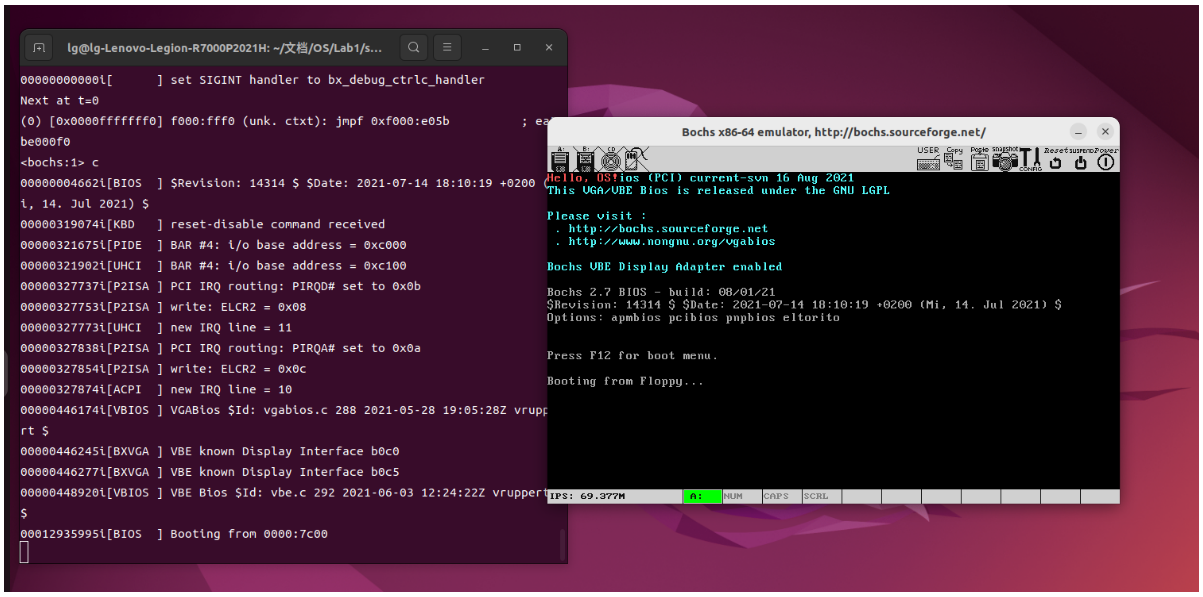
1.1 代码

```
;boot.asm
    org 07c00h
    mov ax, cs
    mov ds, ax
    mov es, ax
    call DispStr
    jmp $
DispStr:
    mov ax, BootMessage
    mov bp, ax
    mov cx, 10
    mov ax, 01301h
    mov bx, 000ch
    mov dl, 0
    int 10h
    ret

BootMessage: db "Hello, OS!"
times 510-($-$$) db 0
dw 0xaa55
```

`boshsrc`

```
megs:32
display_library: sdl2
floppya: 1_44=a.img, status=inserted
boot: floppy
romimage: file=$BXSHARE/BIOS-bochs-legacy
```

1.2 运行截图


lg@lg-Lenovo-Legion-R7000P2021H: ~/文档/OS/Lab1/s...
Bochs x86-64 emulator, http://bochs.sourceforge.net/
USER
Copy
Paste
snapshot
CONFIG
Reset
SUSPEND
Power
IPS: 69.377M
A:
NUM
CAPS
SCRL

```
00000000000i[      ] set SIGINT handler to bx_debug_ctrlc_handler
Next at t=0
(0) [0x0000fffffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b         ; ea
be000f0
<bochs:1> c
00000004662i[BIOS  ] $Revision: 14314 $ $Date: 2021-07-14 18:10:19 +0200 (
i, 14. Jul 2021) $
00000319074i[KBD   ] reset-disable command received
00000321675i[PIDE  ] BAR #4: i/o base address = 0xc000
00000321902i[UHCI  ] BAR #4: i/o base address = 0xc100
00000327737i[P2ISA ] PCI IRQ routing: PIRQD# set to 0x0b
00000327753i[P2ISA ] write: ELCR2 = 0x08
00000327773i[UHCI  ] new IRQ line = 11
00000327838i[P2ISA ] PCI IRQ routing: PIRQA# set to 0x0a
00000327854i[P2ISA ] write: ELCR2 = 0x0c
00000327874i[ACPI  ] new IRQ line = 10
00000446174i[VBIOS ] VGABios $Id: vgabios.c 288 2021-05-28 19:05:28Z vrupp
rt $
00000446245i[BXVGA ] VBE known Display Interface b0c0
00000446277i[BXVGA ] VBE known Display Interface b0c5
00000448920i[VBIOS ] VBE Bios $Id: vbe.c 292 2021-06-03 12:24:22Z vruppert
$
00012935995i[BIOS  ] Booting from 0000:7c00
```

```
Hello, OS!ios (PCI) current-svn 16 Aug 2021
This VGA/VBE Bios is released under the GNU LGPL

Please visit :
 . http://bochs.sourceforge.net
 . http://www.nongnu.org/vgabios

Bochs VBE Display Adapter enabled

Bochs 2.7 BIOS - build: 08/01/21
$Revision: 14314 $ $Date: 2021-07-14 18:10:19 +0200 (Mi, 14. Jul 2021) $
Options: apmbios pcibios pnpbios eltorito


Press F12 for boot menu.

Booting from Floppy...
```

2、实验任务二：进制转换

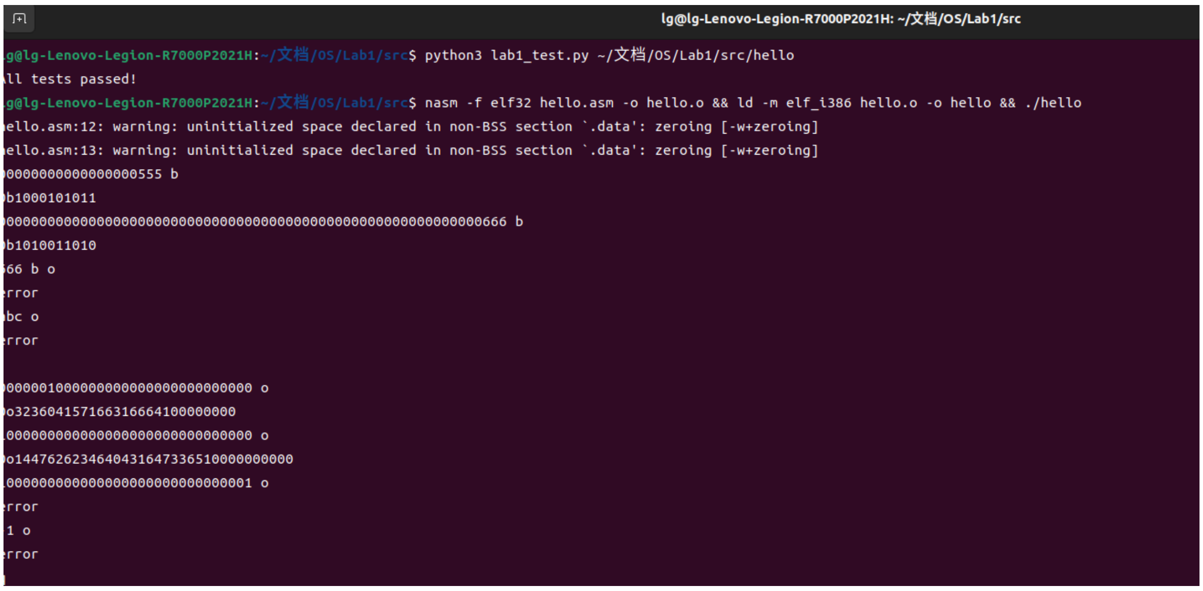
2.1 代码

```
;hello.asm
;; nasm -f elf32 hello.asm -o hello.o && ld -m elf_i386 hello.o -o hello &&
./hello
SECTION .bss
input:  resb 256


SECTION .data
    msginput: db "Please input:", 0h ;输入提示
    msgerror: db "error", 0h         ;error
    msg0x:    db "0x",0h
    msg0b:    db "0b",0h
    msg0o:    db "0o",0h
    num: resb 256 ;存储原数字
    res: resb 256 ;存储答案
    calcbase:  db 0h         ;存储基数
    myflag:    db 0h         ;状态机
    ismaximum: db 0h         ;判断是否等于1e30
    ;input:    db "11 o", 0h




SECTION .text
global  _start

_start:
    call read        ;输入，整行存储于input中
    call solve_input

    ;push eax
    ;mov  eax,          myflag
    ;add  byte[myflag], 48
    ;call charprint
    ;sub  byte[myflag], 48
    ;pop  eax

    cmp byte[myflag], 0
    jz  _start
    cmp byte[myflag], 1
    jz  printerror
    cmp byte[myflag], 2
    jz  printerror
    cmp byte[myflag], 3
    jz  check_num
    cmp byte[myflag], 4
    jz  quit

    cmp byte[myflag], 5
    jz  printerror

solve_input: ;处理输入，提取原数num，目标进制base
```

```
mov eax,          input
mov byte[myflag], 0

nextchar:
    cmp byte [eax], 0
    jz  finished
    cmp byte [eax], 10
    jz  finished
    cmp byte[eax],  13
    jz  finished

    ;小小的调试
    ;push eax
    ;call charprint
    ;pop eax

    ;push eax
    ;add byte[myflag], 48
    ;mov eax, myflag
    ;call charprint
    ;sub byte[myflag], 48
    ;pop  eax


    mov bl, byte[eax] ;ebx存储当前字符
    inc eax

    cmp byte[myflag], 4
    je  solve4
    cmp byte[myflag], 3
    je  solve3
    cmp byte[myflag], 2
    je  solve2
    cmp byte[myflag], 1
    je  solve1
    cmp byte[myflag], 0
    je  solve0


solve0:
    cmp bl, 32
    je  nextchar
    cmp bl, 113   ;等于'q'
    je  set4
    cmp bl, 48
    jl  finisherr
    cmp bl, 57
    jg  finisherr
    jmp set1

set1:
    mov byte[myflag], 1

    mov ecx, num
    mov byte[ecx], bl
```

```
        inc ecx
        jmp nextchar

    solve1:
        cmp bl,        32
        je  set2
        cmp bl,        48
        jl  finisherr
        cmp bl,        57
        jg  finisherr
        mov byte[ecx], bl
        inc ecx
        jmp nextchar

    set2:
        mov byte[myflag], 2
        mov byte[ecx],    0
        jmp nextchar

    solve2:
        cmp bl, 32
        je  nextchar
        cmp bl, 98
        je  set3b
        cmp bl, 111
        je  set3o
        cmp bl, 104
        je  set3h
        jmp finisherr

    set3b:
        mov byte[calcbase], 2
        mov byte[myflag],   3
        jmp nextchar
    set3o:
        mov byte[calcbase], 8
        mov byte[myflag],   3
        jmp nextchar

    set3h:
        mov byte[calcbase], 16
        mov byte[myflag],   3
        jmp nextchar

    solve3:
        cmp bl, 32
        je  nextchar
        jmp finisherr



    set4:
        mov byte[myflag], 4
        jmp nextchar

    solve4:
        cmp bl, 32
        je  nextchar
        jmp finisherr

    finisherr:
        mov byte[myflag], 5
        jmp finished

    finished:
        ret

solve_num: ;处理num前导 0
    push eax
    push ecx
    mov  eax, num
    mov  ecx, num

    solve_num_loop1:
        cmp byte[eax], 48
        jne solve_num_loop2
        inc eax
        jmp solve_num_loop1

    solve_num_loop2:
        ;byte[ecx]=byte[eax]
        push eax
        mov  al,        byte[eax]
        mov  byte[ecx], al
        pop  eax

        cmp byte[ecx], 0
        jz  end_solve_num
        inc ecx
        inc eax
        jmp solve_num_loop2
    end_solve_num:
        pop ecx
        pop eax
        ret

check_num: ;判断数落在正确区间内

    call solve_num            ;去0
    mov  byte[ismaximum], 1
    mov  eax,             num

    mov bl, byte[eax]
    mov bh, 0          ;index
    cmp bl, 0
    jz  calc_res
    cmp bl, 49
    jne no_one
    inc eax
    inc bh
    jmp loop_check_num

    loop_check_num:
        cmp byte[eax], 0
        jz  end_loop_check_num1
        cmp byte[eax], 48
        jne no_zero
        inc eax
        inc bh
        jmp loop_check_num

    no_one:
        mov byte[ismaximum], 0
        inc eax
        inc bh
        jmp loop_check_num

    no_zero:
        mov byte[ismaximum], 0
        inc eax
        inc bh
        jmp loop_check_num

    end_loop_check_num1:
        cmp bh, 31
        jg  printerror
        cmp bh, 31
        je  end_loop_check_num2
        jmp calc_res

    end_loop_check_num2:
        cmp byte[ismaximum], 1
        je  calc_res
        jmp printerror



calc_res: ;获取答案，高精除法，模拟长除法，edx作为余数寄存器，eax作为被除数寄存器，ebx作为num指针，ecx作为res指针
    mov ecx, res
    begin_calc_loop:
        call solve_num
        ; 判断被除数为0，0则结束
        mov  eax,       0
        mov  edx,       0
        mov  ebx,       num
        cmp  byte[ebx], 0
        jne  calc_loop      ;若被除数不为0,则calc_loop模拟长除法
        mov  byte[ecx], 0
        jmp  solveres

    calc_loop:
        cmp   byte[ebx], 0
        jz    end_calc_loop             ;遍历完毕，结束循环
        mov   eax,       edx
        imul  eax,       10
        push  ebx
        movzx ebx,       byte[ebx]
        add   eax,       ebx
        sub   eax,       48
        movzx ebx,       byte[calcbase]
        div   ebx
        pop   ebx
        mov   byte[ebx], al
        add   byte[ebx], 48
        inc   ebx
        jmp   calc_loop

    end_calc_loop:
        call get_digit
        mov  byte[ecx], dl
        inc  ecx
        jmp  begin_calc_loop


solveres: ;翻转并去除前导0
    mov eax, res
    mov ecx, res
    loop_reverse_begin:
        cmp byte[ecx], 0
        jz  loop_reverse_then
        inc ecx
        jmp loop_reverse_begin
    loop_reverse_then:
        dec ecx
    loop_reverse:
        cmp eax,       ecx
        jnl end_loop_reverse
        mov bh,        byte[eax]
        mov bl,        byte[ecx]
        mov byte[eax], bl
        mov byte[ecx], bh
        inc eax
        dec ecx
        jmp loop_reverse

    end_loop_reverse:
        mov eax, res
        mov ecx, res
    loop_nozero1:
        cmp byte[ecx], 48
        jne loop_nozero2
        inc ecx
        jmp loop_nozero1
    loop_nozero2:
        mov bl,        byte[ecx]
        mov byte[eax], bl
        cmp byte[ecx], 0
        jz  work_nozero1
        inc ecx
        inc eax
        jmp loop_nozero2
    work_nozero1:
        mov eax,       res
        cmp byte[eax], 0
        jz  work_nozero2
        jmp end_solveres
    work_nozero2:
        mov byte[eax], 48
        inc eax
        mov byte[eax], 0
    end_solveres:
        jmp printres

printres:
    push eax
    cmp  byte[calcbase], 2
    jz   output0b
    cmp  byte[calcbase], 8
    jz   output0o
    cmp  byte[calcbase], 16
    jz   output0x


    outputres:
    mov  eax, res
    call strprintln
    pop  eax
    jmp  _start

output0b:
    mov  eax, msg0b
    call strprint
    jmp  outputres
output0o:
    mov  eax, msg0o
    call strprint
    jmp  outputres

output0x:
    mov  eax, msg0x
    call strprint
    jmp  outputres




get_digit: ;获取数字(edx)对应字符
    cmp edx, 10
    jl  isdigit
    sub edx, 10
    add edx, 97
    ret
    isdigit:
        add edx, 48
    ret

strlen: ;求字串长
    push ebx
    mov  ebx, eax
    strlennextchar:
        cmp byte [eax], 0
        jz  strlenfinished
        inc eax
        jmp strlennextchar
    strlenfinished:
        sub eax, ebx
        pop ebx
        ret

read:
    push eax
    ;mov  eax, msginput
    ;call strprint
    push ebx
    push ecx
    push edx
    mov  edx, 255
    mov  ecx, input
    mov  ebx, 0
    mov  eax, 3
    int  80h
    pop  edx
    pop  ecx
    pop  ebx
    pop  eax
    ret

charprint: ;输出字符
    push edx
    push ecx
    push ebx
    mov  ecx, eax
    mov  edx, 1
    mov  ebx, 1
    mov  eax, 4
    int  80h
    pop  ebx
    pop  ecx
    pop  edx
    ret



strprint: ;输出字串
    push edx
    push ecx
    push ebx
    push eax
    push eax
    call strlen
    mov  edx, eax
    pop  eax
    mov  ecx, eax
    mov  ebx, 1
    mov  eax, 4
    int  80h
    pop  eax
    pop  ebx
    pop  ecx
    pop  edx
    ret

strprintln: ;输出字串带换行
    call strprint
    push eax
    mov  eax, 0ah
    push eax
    mov  eax, esp
    call strprint
    pop  eax
    pop  eax
    ret

printerror:
    push eax
    mov  eax, msgerror
    call strprintln
    pop  eax
    jmp  _start

quit: ;退出
    mov ebx, 0 ;正常退出
    mov eax, 1
    int 80h    ;调用 SYS_EXIT, 正常退出
    ret

printnum:
    push eax
    mov  eax, num
    call strprintln
    pop  eax
    ret
```

2.2 运行截图



```
lg@lg-Lenovo-Legion-R7000P2021H: ~/文档/OS/Lab1/src
g@lg-Lenovo-Legion-R7000P2021H:~/文档/OS/Lab1/src$ python3 lab1_test.py ~/文档/OS/Lab1/src/hello
ll tests passed!
g@lg-Lenovo-Legion-R7000P2021H:~/文档/OS/Lab1/src$ nasm -f elf32 hello.asm -o hello.o && ld -m elf_i386 hello.o -o hello && ./hello
ello.asm:12: warning: uninitialized space declared in non-BSS section `.data': zeroing [-w+zeroing]
ello.asm:13: warning: uninitialized space declared in non-BSS section `.data': zeroing [-w+zeroing]
0000000000000000555 b
b1000101011
00000000000000000000000000000000000000000000000000000000000000666 b
b1010011010
66 b o
rror
bc o
rror

00000100000000000000000000000000 o
o3236041571663166641000000000
10000000000000000000000000000000 o
o14476262346404316473365100000000000
10000000000000000000000000000001 o
rror
1 o
rror
```