Shellcode

S9.1. 32 位 Linux shellcode 在触发 `execve()` 系统调用前，需要准备 `eax`、`ebx`、`ecx` 和 `edx`。这四个寄存器应分别包含什么？

S9.2. 使用基于栈的方法时，需要把命令字符串放入内存，并将其地址保存到 `ebx`。编写一段 32 位代码，把字符串“aaaabbbbccccdddd”压入栈中，再令 `ebx` 指向它。

S9.3. 使用基于栈的方法时，还要在内存中构造参数数组 `argv[]`，并让 `ecx` 指向它。编写一段 32 位代码构造下列数组。

```
argv[0] = 0x11111111
argv[1] = 0x22222222
argv[2] = 0x33333333
argv[3] = 0x00000000
```

S9.4. 编写 shellcode 时，基于代码段的方法与基于栈的方法主要有什么区别？

S9.5. 下列 shellcode 不完整。请用实际数字替换所有 `*`，并为带圈编号的每一行写出用途。不能只复述指令语义，还要说明它在构造 `execve()` 参数中的作用。

```
section .text
  global _start
    _start:
        BITS 32
        jmp short two
    one:
        pop ebx              ①
        xor eax, eax
        mov [ebx+*],  al     ②
        mov [ebx+*],  ebx    ③
        mov [ebx+*], eax     ④
        lea ecx, [ebx+*]     ⑤
        xor edx, edx
        mov al,  0x0b
        int 0x80
     two:
        call one
```

```
        db '/bin/shabcde'    ⑥
        db 'AAAA'            ⑦
        db 'BBBB'            ⑧
```

S9.6. 在下列 32 位 shellcode 中，用偏移量替换问号，用寄存器名替换星号，并为每一条补全的指令添加注释。

```
section .text
  global _start
    _start:
        BITS 32
        jmp short two
    one:
        pop edx
        xor eax, eax
        mov [*+?],  al
        lea ebx, [*+?]
        mov [*+?], eax
        mov [*+?], ebx
        mov ecx, *
        xor *, *
        mov al,  0x0b
        int 0x80
     two:
        call one
        db 'AAAA'
        db 'BBBB'
        db '/bin/sh*'
```

S9.7. 下列 shellcode 的目标是执行命令 `"/bin/rm -rf *"`。请根据给出的数据布局补全代码，并写出清晰注释。

```
  _start:
      jmp short two
  one:
      ... add code here ...

      mov al,  0x0b   ; invoke execve() system call
      int 0x80
  two:
      call one
```

```
    db 'abcd/bin/rmab-rf****'
    db 'AAAABBBBCCCCDDDDEEEEFFFFGGGG'
```

S9.8. 为什么 shellcode 的机器码通常不能包含零字节？

S9.9. 列出三种在 shellcode 中消除零字节的常见方法。

S9.10. 希望在栈上存放以零结尾的字符串“ab”，但机器码中不能直接含零字节。（1）补全小端机器代码；（2）补全大端机器代码。

```
mov ecx, "ab**"
... (missing code) ...
push ecx
```

S9.11. ★

把 `0xAA00BB00` 构造到 `eax` 中，要求最终机器码不含零字节。