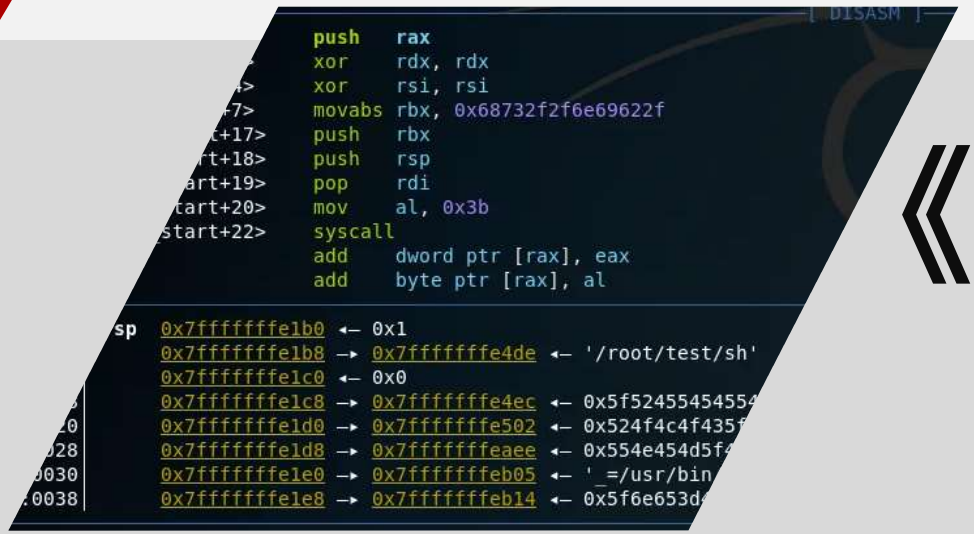
2023年春季学期



《计算机系统》

汇编进阶

《计算机系统》课程教学组

地址传送
地址计算

lea (load effective address) 指令

- lea 指令是mov 指令的变形
- lea 指令形式看上去是从存储器读数据到寄存器
- 但实际并没有引用存储器
- 而是将有效地址写入目的操作数（必须是寄存器）
- leal S，D 表示D ← &S

LEA指令用途

1) 地址计算 / 地址传送

若寄存器 %edx 的值为 x，那么

leal 7（%edx，%edx，4），%eax

表示：寄存器%eax的值为 5x+7

lea VS. mov

- leal 8(%edi), %eax
- ==> %eax = 8 + %edi

- movl 8(%edi), %eax
- ==> %eax = M[8 + %edi]

- **演示讲解 1012.s**

LEA指令用途

2）用来执行简单的算术操作

```
int scale（int x , int y, int z）
{
  int t= 8*z+2*y+5*x ;
  return t;
}
```

4*x+x=5*x

2*y+5*x

8*z+2*y+5*x

设若x in %edi, y in %esi, z in %edx
反汇编后得到汇编代码片段如下：

```
scale:

   leal   (%edi, %edi, 4) , %eax
   leal   (%eax, %esi, 2) , %eax
   leal   (%eax, %edx, 8) , %eax
   ret
```

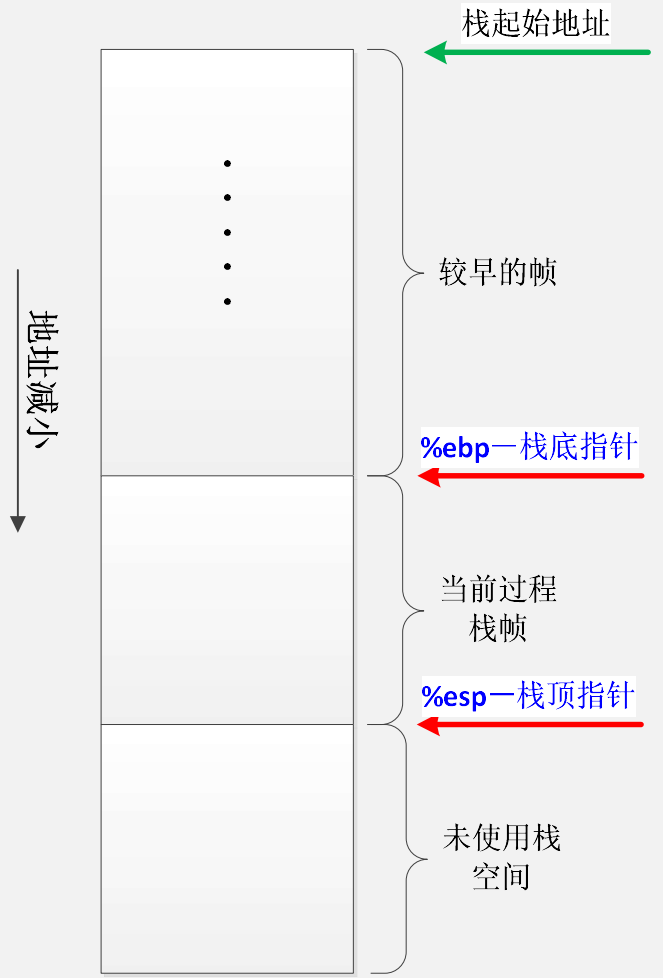
请填写左边代码空白处！

因为比例因子只能是1,2,4,8，在LEA的独立电路中用移位寄存器可以简单高效地实现此类计算！

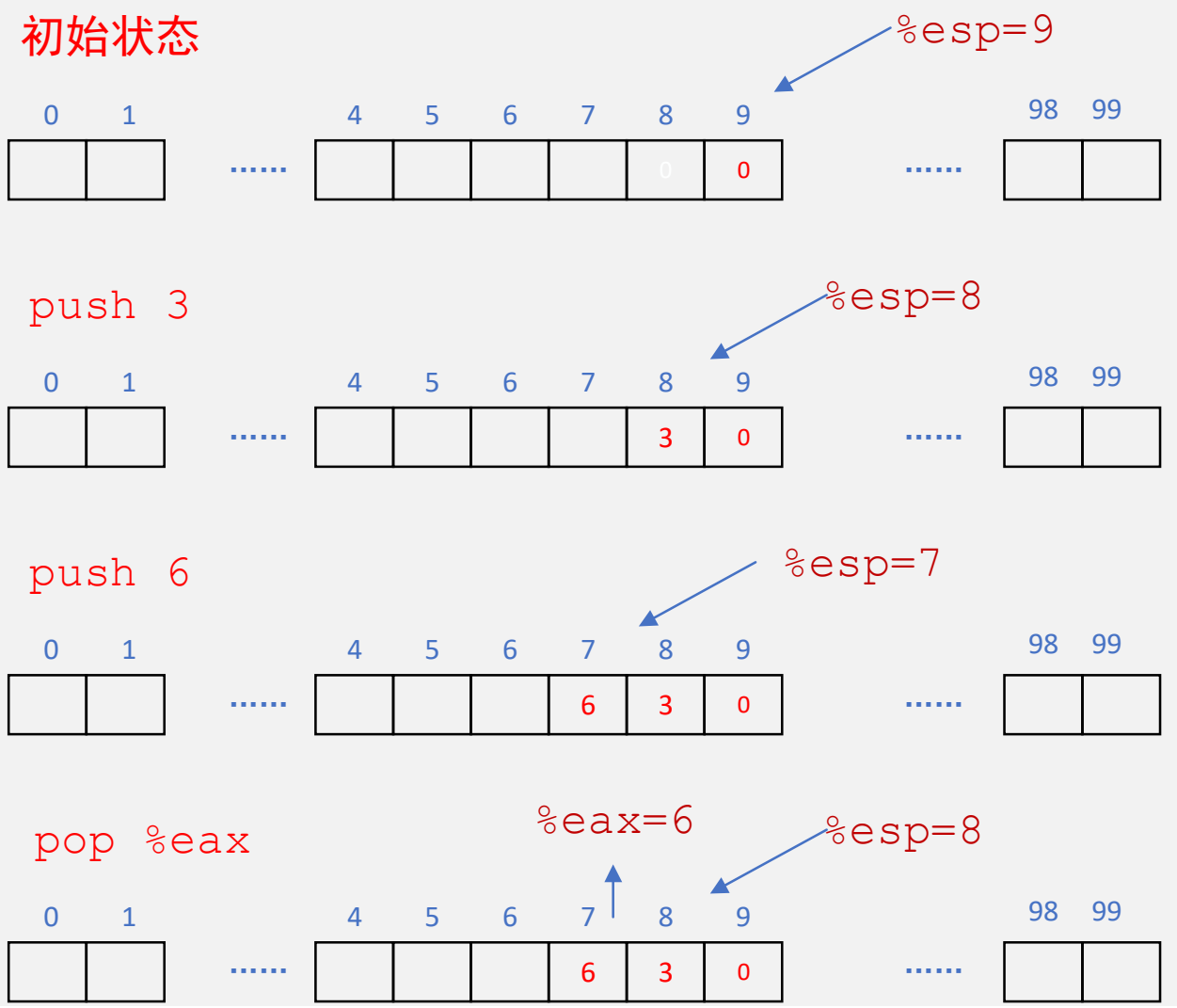
函数的栈帧

每一个函数或过程在执行时，都需要在内存中分配一个空间来保存运行时数据，这个空间由于是采用栈的方式进行操作，所以也称为**栈帧**。

- 当前函数或过程的栈顶地址保存在%esp中，栈底地址保存在%ebp中；
- 栈是向“下”增长的，或者说是向地址0x0处增加的，因此%esp中的值小于或等于%ebp中的值；
- 栈帧是内存中一段连续的内存空间；
- 被调用者的栈帧紧挨着调用者的栈帧；

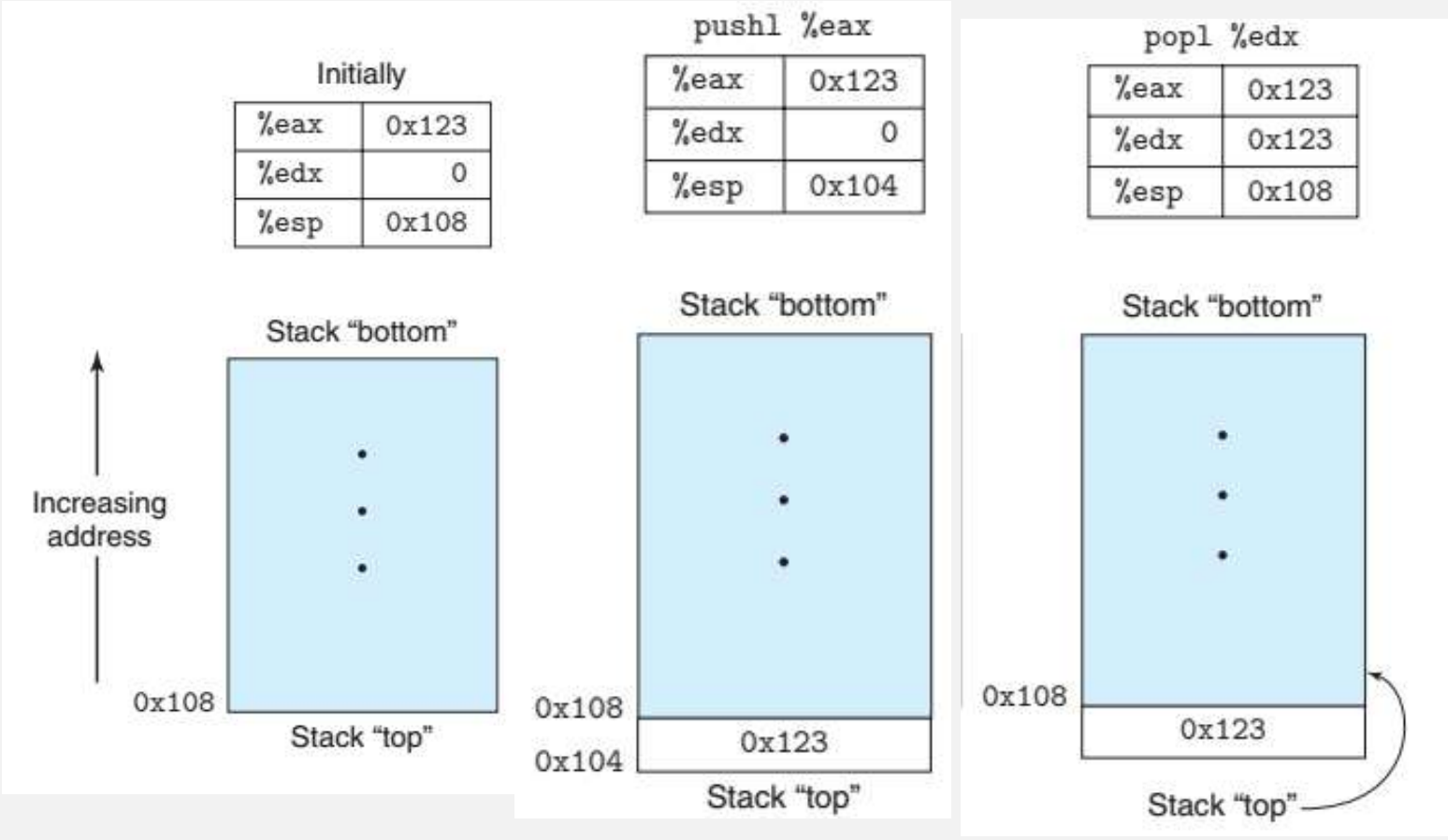


栈顶是朝着低地址方向
（栈是向着零地址方向增长）



栈操作示例

0x123为4字节整形数据



栈操作指令

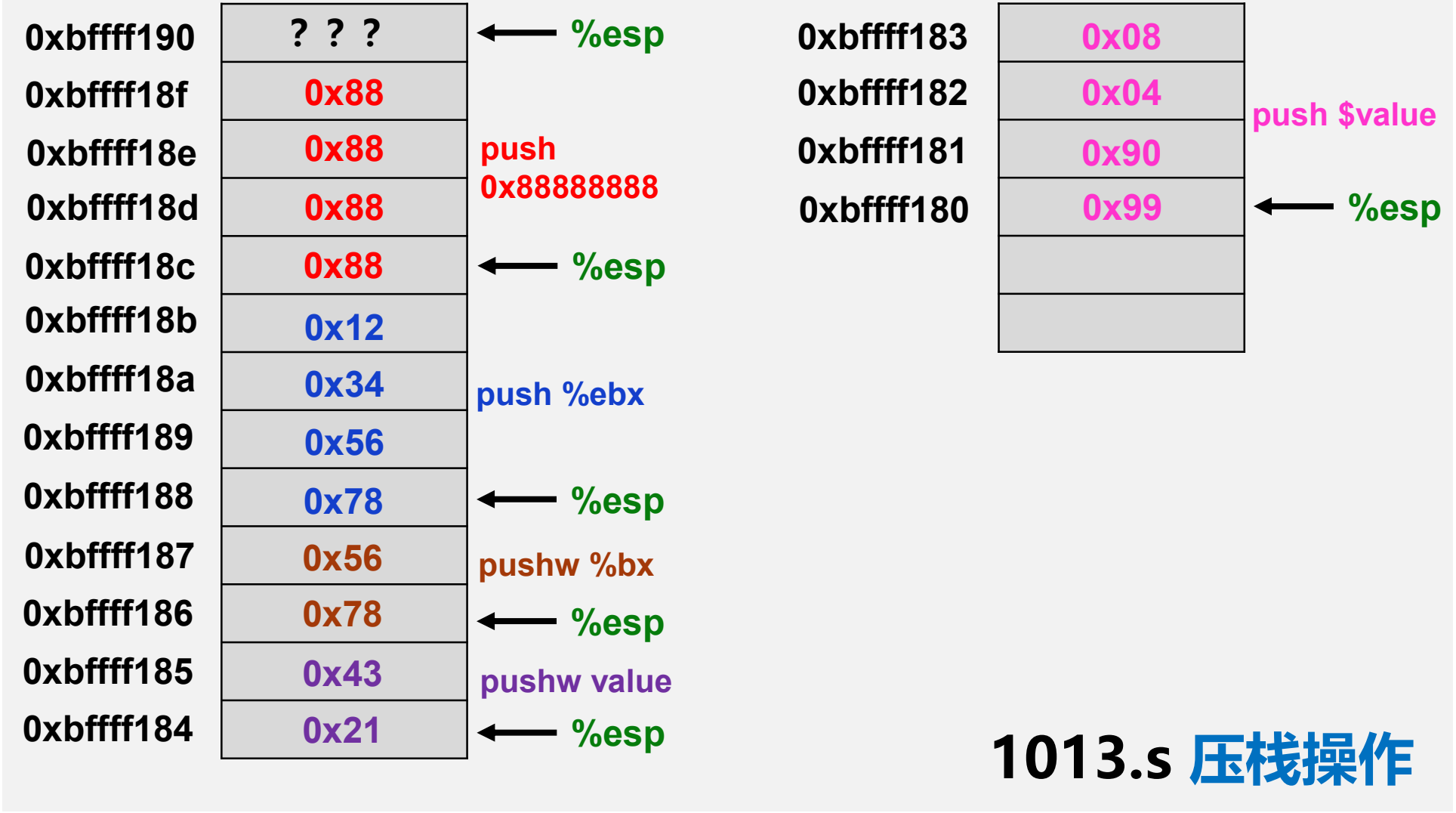
在每个程序所分配的内存中，划分出一段连续的区域，作为栈空间（1013.s）

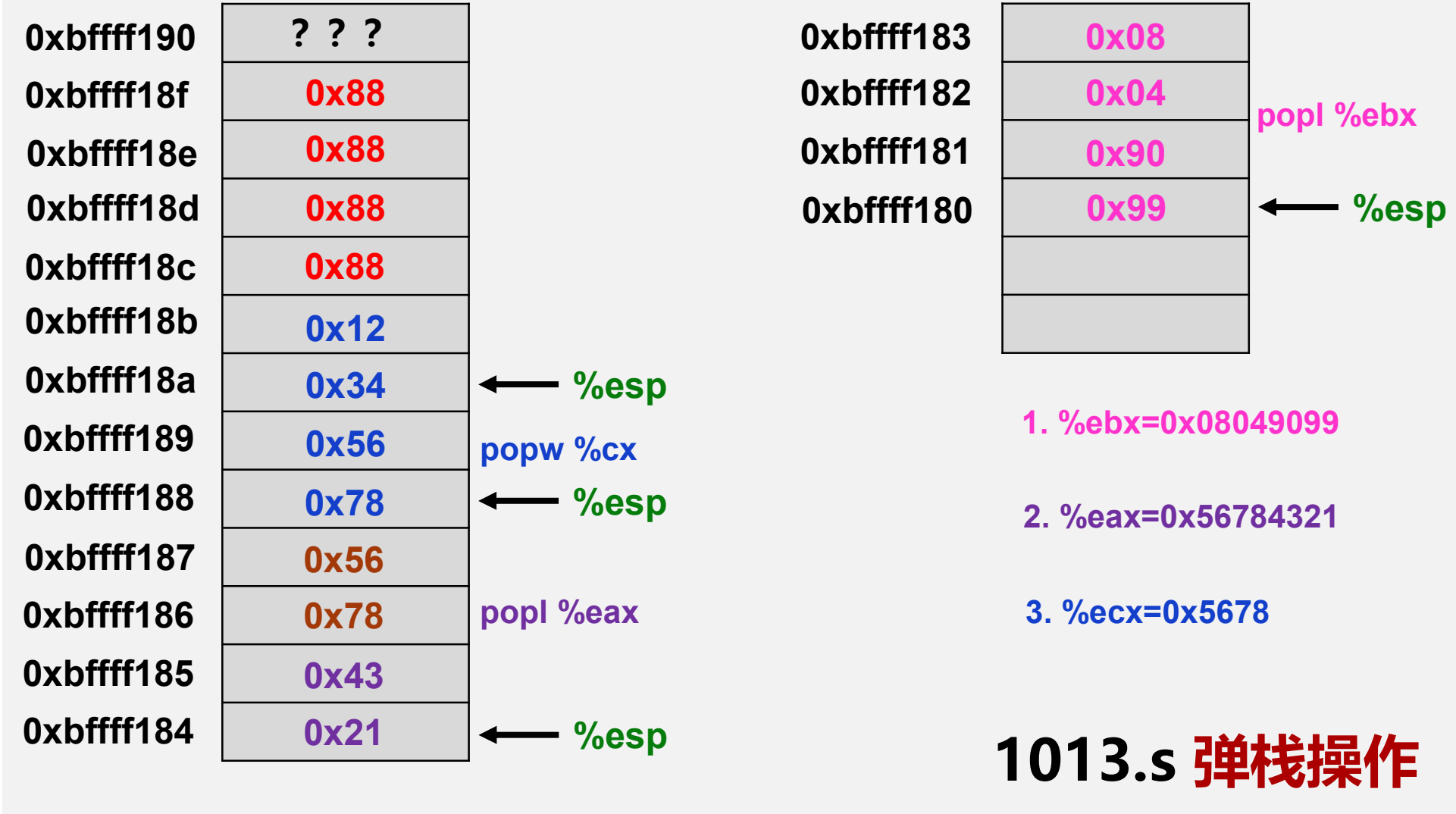
栈：先进后出

栈顶指针：保存在%esp寄存器中

压栈：push

出栈：pop


0xbffff190
? ? ?
%esp
0xbffff18f
0x88
0xbffff18e
0x88
push 0x88888888
0xbffff18d
0x88
0xbffff18c
0x88
%esp
0xbffff18b
0x12
0xbffff18a
0x34
push %ebx
0xbffff189
0x56
0xbffff188
0x78
%esp
0xbffff187
0x56
pushw %bx
0xbffff186
0x78
%esp
0xbffff185
0x43
pushw value
0xbffff184
0x21
%esp
0xbffff183
0x08
0xbffff182
0x04
push $value
0xbffff181
0x90
0xbffff180
0x99
%esp
1013.s 压栈操作


0xbffff190
? ? ?
0xbffff18f
0x88
0xbffff18e
0x88
0xbffff18d
0x88
0xbffff18c
0x88
0xbffff18b
0x12
0xbffff18a
0x34
%esp
0xbffff189
0x56
popw %cx
0xbffff188
0x78
%esp
0xbffff187
0x56
0xbffff186
0x78
popl %eax
0xbffff185
0x43
0xbffff184
0x21
%esp
0xbffff183
0x08
0xbffff182
0x04
popl %ebx
0xbffff181
0x90
0xbffff180
0x99
%esp
1. %ebx=0x08049099
2. %eax=0x56784321
3. %ecx=0x5678
1013.s 弹栈操作

算术逻辑操作指令

一般算术/逻辑指令

incl D	加1操作	decl D	减1操作	SHR k, D	逻辑右移
negl D	取负	notl D	取反	SHL k, D	逻辑左移
addl S,D	加法	subl S,D	减法	SHA k, D	算术右移
imull S,D	乘	xorl S,D	异或	SAL k, D	算术左移
orl S,D	或	andl S,D	与		

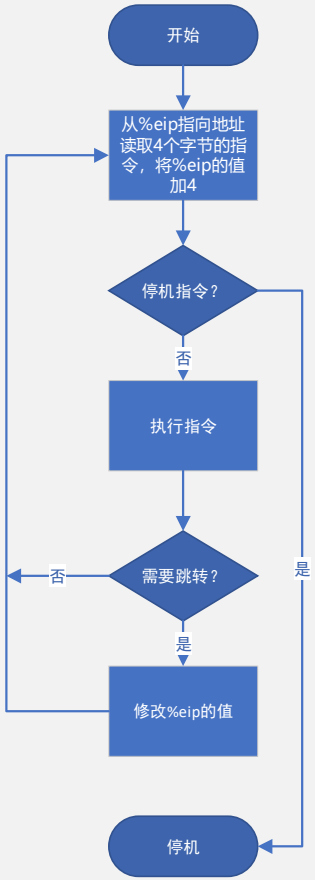
特殊算术指令

imull S	有符号乘法——将S与%eax中的值相乘，64位结果的高32位放%edx，低32位放%eax
mull S	无符号乘法——将S与%eax中的值相乘，64位结果的高32位放%edx，低32位放%eax
clt S	将%eax中的值按符号位扩展的方式转换为64位值，高32位放%edx，低32位放%eax
idivl S	有符号除法——R[%edx] = R[%edx]:R[%eax] mod S; R[%eax] = R[%edx]:R[%eax] / S;
divl S	无符号除法——R[%edx] = R[%edx]:R[%eax] mod S; R[%eax] = R[%edx]:R[%eax] / S;

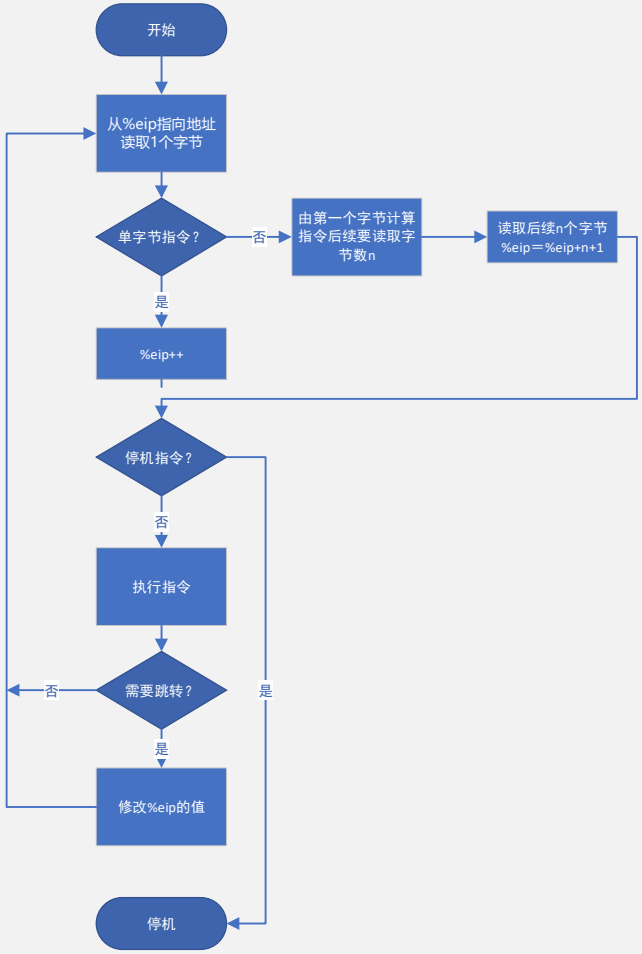
跳转指令

%eip - - 存放下一条要执行指令的地址

定长指令集（假设为32位）



变长指令集



无条件跳转

JMP 指令

格式：　jmp Label

```
804805a:       eb 05                   jmp    8048061 <exit>
804805c:       b8 05 00 00 00          mov    $0x5,%eax

08048061 <exit>:
8048061:       b8 01 00 00 00          mov    $0x1,%eax
```

1014.s

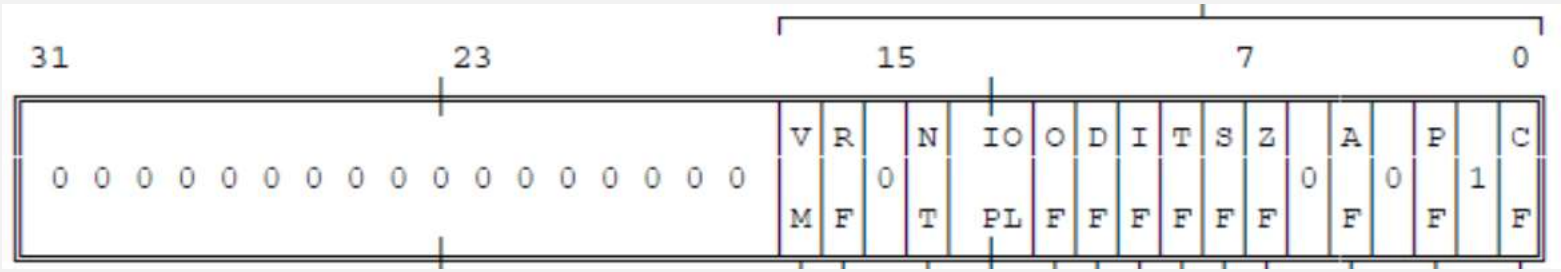
详见演示

这里的05，是将指下一条指令（mov $5, %eax）的地址（即当前PC所存地址）再加上05，就能得到跳转目的地<exit>的地址 （教材pp.128）。

0x804805c + 5 = 0x8048061 → exit所在的地址

条件跳转指令

条件码在%eflags寄存器中



SF	符号位	第7位
ZF	零标志位	第6位
CF	进位	第0位
OF	溢出	第11位

```
c code

if (x>y)
        A;
else
        B;
```

汇编执行思路

- 做减法x-y或比较x与y(cmp y,x);
- 根据x-y的结果，设置CF,OF,SF,ZF各个位;
- 根据程序需求来判断用到的位。

条件码相关知识

- **inc**和**dec**指令不影响进位标志位，而`add $1,%eax`与`sub $1,%eax`等指令会影响进位标志位

- **cmp**指令是对两个数做减法，但不保留结果，仅根据结果设置标志位

条件码相关知识

- **test**指令对两个操作数做逻辑与运算，但不保留结果

```
testl    $0x4,%eax
         #0x4=00000000 00000000 00000000 00000100
jnz      *** #如果eax的倒数第三个bit为1，则跳转
testl    %ecx, %ecx
jz       **     #如果ecx为零，则跳转
```

- 对于CF标志位，有三条专门的指令

clc：将CF标志位清零
stc：将CF标志位设置为1
cmc：将CF标志位置反

循环指令

- 循环可以通过跳转指令来实现，也可以利用loop指令来实现
- 计算一个整数数组中数字的和
- 1019.s

2023年春季学期

下一节：信息表达

《计算机系统》课程教学组