（由计科1808班刘宏源同学整理）

如何一步步将程序变为机器语言

**一.hello.c=>hello.i(源程序，文本)**

指令：**gcc -E hello.c -o hello.i**

-E:表示预编译

1.编译C源代码中的包含的头文件

2.预编译器(cpp)根据以字符#开头的命令，修改原始C程序

如hello.c中的#include<stdio.h>指令告诉预处理器读系统头文件stdio.h的内容，并将其直接插入到程序文本中去

结果就得到另一个C程序，通常是以.i作为文件扩展名的

**二.hello.i=>hello.s(汇编语言程序，文本)**

指令：**gcc -S hello.i -o hello.s**

1.gcc检查代码的(1.规范性，2.是否有语法错误，3.确定代码实际要做的工作)

2.gcc将代码翻译成汇编语言

**三.hello.s=>hello.o（可重定位目标程序，二进制，hello.o)**

指令：**gcc -c hello.s -o hello.o**

1.汇编阶段将编译阶段生成的.s文件转成二进制目标代码

**四.解释.o文件**

指令：objdump -d hello.o

解释.o文件

**五.链接过程 (可执行目标程序，二进制 hello)**

指令：gcc hello.o -o hello

|  |  |  |  |
| --- | --- | --- | --- |
| xxxx | xxxx | xxxx | xxxx |
|  |  | ah | al |
|  |  | ax | |
| eax | | | |

**gcc -S 1017.c -o 1017.s**

将c程序变成汇编代码

**gcc -g 1020.c -o 1020**

将c程序变成可执行的二进制文件

**gcc -O2 -g -o p main.c swap.c**

**./p**

编译器对多个文件进行翻译和链接

**b \*\_main**

**b main**

把断点加在main上

**list (l)**

查看c语言代码

**disassem**

查看汇编代码

**ni**

执行下一条汇编代码

**gcc 1017.c**

编译程序

**./a.out**

运行程序

**as -g 1005.s -o 1005.o //汇编小测**

将1005.s(汇编代码)变成二进制代码

**objdump -d 1005.o**

解释.o文件

**ld 1005.o -o 1005 //汇编小测**

把文件变成可执行的二进制文件

**./1005**

执行文件

**gdb -q 1005**

调试文件

**b \*\_start+1**

在start加1的一行停住，加个断点

**r**

运行

**info reg(i r)**

查看寄存器的信息

**n**

执行下一条语句

movl:传送32位的长字值

movw:传送16位的字值

movb:传送8位的字值

mov $4, %eax ：将数值4传给eax

mov %eax,%ebx ：将%eax的值传给%ebx

存入数据先低位，后高位

取出数据先高位，后低位

小端法：高的放高位

**x/8bx 0x08048056**

查看8 bit 16进制的内存内容

(b-单字节，h-双字节，w-四字节，g-八字节。

x 按十六进制格式显示变量。d 按十进制格式显示变量。

u按十六进制格式显示无符号整型。o按八进制格式显示变量。

t 按二进制格式显示变量。a 按十六进制显示变量。

c 按字符格式显示变量。f 按浮点数格式显示变量。)

movl $1,%eax #立即数寻址 -1005.s

movl %ebx,%eax #寄存器寻址 -1006.s

movl 0x08048054,%eax #绝对寻址 -1007.s

movl (%ebx),%eax #间接寻址 -1008.s

movl 0x8(%ebx),%eax #基址偏移量寻址 ebx+8

movl (%ebx,%edx),%eax #变址寻址 -1009.s ebx+edx

movl 0x8(%ebx,%edx),%eax #变址基址寻址 ebx+edx+8

movl (%ebx,%ecx,0x2),%eax #比例变址寻址 -1010.s

ebx+ecx\*2

movl 0x8(%ebx,%ecx,0x2),%eax #比例变址基址寻址

ebx+ecx\*2+8

**把值送到内存里面去**

.section.data

定义数据段：

value1:

.int 1

value2:

.short 2

value3:

.byte 3

**p value1**

查看值

**p &value1**

查看地址

**x/8bx $esp**

查看栈顶

**p $ebx**

查看ebx的地址

**p/x $ebx**

以十六进制查看ebx的地址

movl value2,%eax 把从value2地址开始4个字节给eax，无关数据类型。

$value1 是value1的地址

获得变量在内存中的地址并传送给寄存器(1012.s)

movl $value1,% 先得到地址，再取值，再给寄存器

lea value1,% 得到地址就给寄存器

**函数的栈帧**

1.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

2.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

3.栈帧是内存中一段连续的内存空间

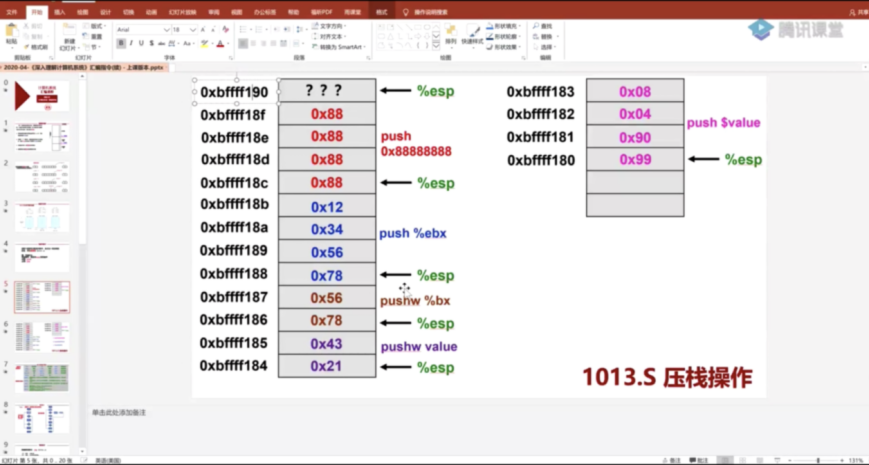
4.被调用者的栈帧紧挨着调用者的栈帧

pushl $0x88888888

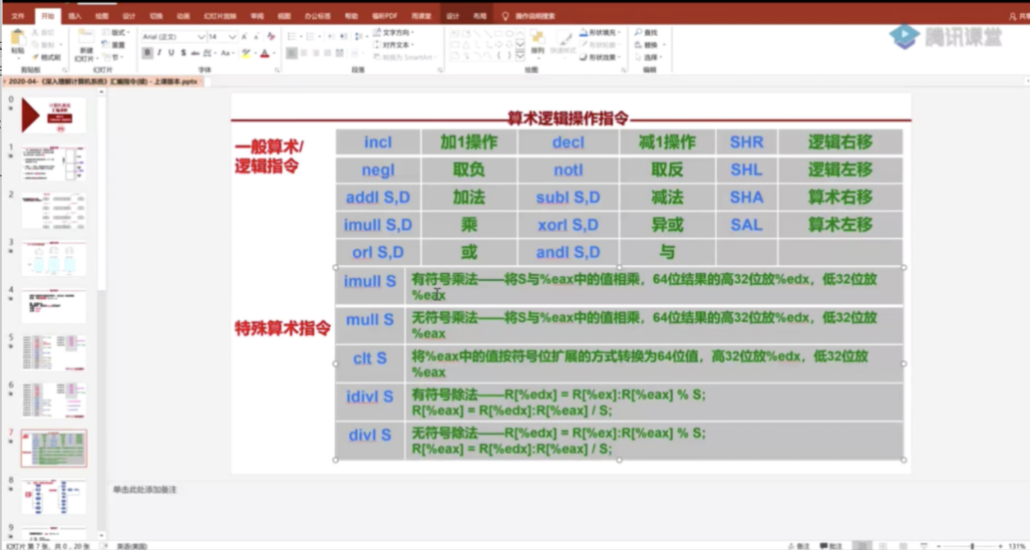
pushw value 把value开始的2个字节压入栈中

每压入一个字节，栈顶元素就往0的方向前进一个字节

popl %ebx取出4个字节的栈顶元素给ebx，栈顶元素向远离0的方向前进4个字节



**算数逻辑操作指令**



**跳转指令**

%eip 存放下一条要执行指令的地址

定长指令集：浪费字节

变长指令集：可加快速度，但需要判断取多少个字节的指令

**p $eip**

查看eip里面存储的内容

直接跳转指令：jmp(1014.s)

1.jmp lable #lable 是一个标号

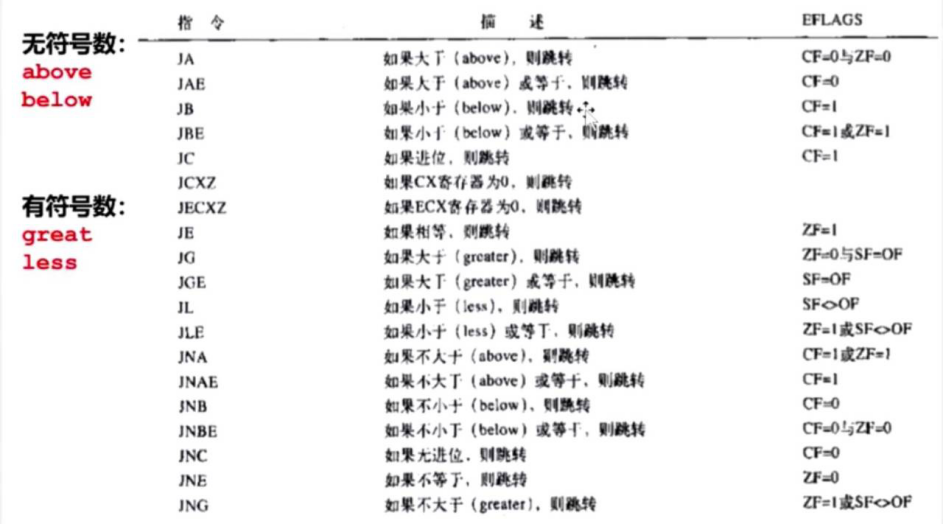
2.jmp 0x0804909a #指令的地址

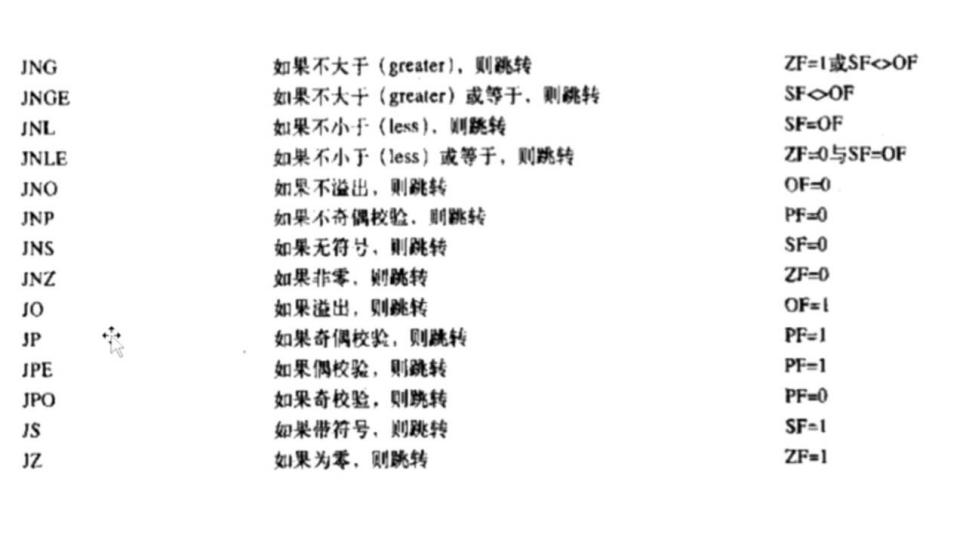
3.jmp \*%eax #%eip跳转到%eax

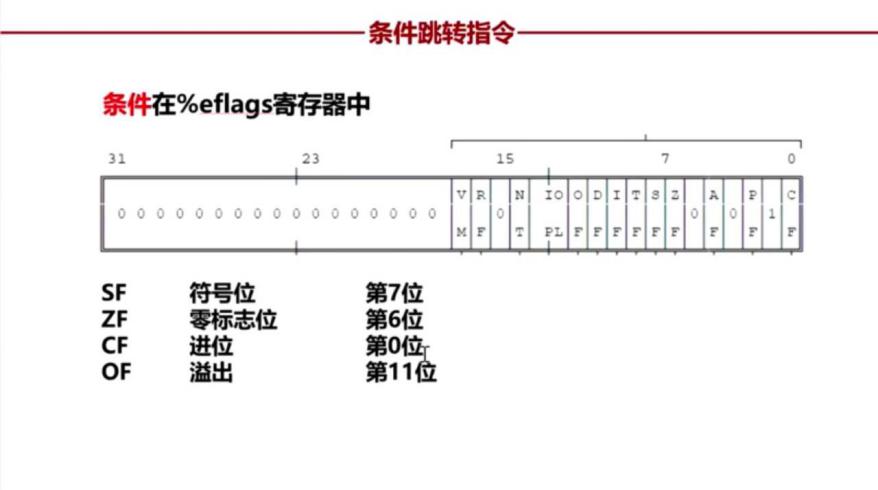
4.jmp \*lable #加\*和不加\*都是可以的

**条件跳转指令**

条件在%eflags寄存器中









cmp %ah,%al #把后面的数减去前面的数，然后根据结果来设置标志位

ja exit #看作无符号数，如果后面的数比前面的数要大，就跳转

jg exit #看作有符号数

cmp 和 ja等指令 是合起来用的

cmpl $0x0,0x18(%esp)

setg %al #如果0x18(%esp)大于$0x0，则al就是1，否则就是0

**p $eflags**

查看标志位，若标志位为1则显现出来，如果为0则不显现