《计算机系统基础》习题课

第三章 程序的表示

2019年10月18日

习题布置

- 第3章习题，第二版课本 pg. 168开始
- 习题 4、5、6、15、17、22、25、28、31、33

- 习题截止时间
  - 2019年10月31日24时，即，11月1日0时

习题

■ 一条机器指令通常由多个字段构成。以下选项中，（ ）不显式地包含在机器指令中。

A. 操作码　　B. 寻址方式　　C

C. 下条指令地址　　D. 寄存器编号

■ 对于运算类指令或传送类指令，通常需要在指令中指出操作数或操作数所在的位置。通常，指令中指出的操作数不可能出现在（ ）中。

A. 指令　　B. 通用寄存器　　D

C. 存储单元　　D. 程序计数器

■ 以下选项中，不属于指令集体系结构名称的是（ ）。

A

A. UNIX　　B. IA-32

C. ARM　　D. MIPS

■ 以下关于IA-32指令寻址方式的叙述中，错误的是（ ）。

A. 操作数可以是指令中的立即数、也可以是通用寄存器或存储单元中的内容

B. 对于寄存器操作数，必须在指令中给出通用寄存器的3位编号

C. 存储器操作数中最复杂的寻址方式是“基址加比例变址加位移”

D. 相对寻址的目标地址为“PC内容加位移”，PC内容指当前正在执行指令的地址

D

■以下关于IA-32指令格式的叙述中，错误的是（ ）。 D

A. 采用变长指令字格式，指令长度从一个字节到十几个字节不等

B. 采用变长操作码，操作码位数可能是5位到十几位不等

C. 指令中指出的位移量和立即数的长度可以是0、1、2或4个字节

D. 指令中给出的操作数所在的通用寄存器的宽度总是32位

■ 以下x87 FPU浮点处理指令系统叙述中，错误的是（ ）。 D

A. 提供8个80位浮点寄存器ST(0)~ST(7)，采用栈结构，栈顶为ST(0)

B. float、double和long double三种类型数据都按80位格式存放在浮点寄存器中

C. float、double和long double型数据存入主存时，分别占32位、64位和96位

D. float和double型数据从主存装入浮点寄存器时有可能发生舍入，造成精度损失

指令集体系结构（ISA）是计算机系统中必不可少的一个抽象层，它是对硬件的抽象，软件通过它所规定的指令系统规范来使用硬件。以下有关ISA的叙述中，错误的是（ ）

B

A. ISA规定了所有指令的集合，包括指令格式和操作类型

B. ISA规定了执行每条指令时所包含的控制信号

C. ISA规定了指令获取操作数的方式，即寻址方式

D. ISA规定了指令的操作数类型、寄存器结构、存储空间大小、编址方式和大端/小端方式

以下关于IA-32的定点寄存器组织的叙述中，错误的是（ ）

A

A. 每个通用寄存器都可作为32位、16位或8位寄存器使用

B. 寄存器EAX/AX/AL称为累加器， ECX/CX/CL称为计数寄存器

C. 寄存器ESP/SP称为栈指针寄存器， EBP/BP称为基址指针寄存器

D. EIP/IP为指令指针寄存器， 即PC；EFLAGS/FLAGS为标志寄存器

IA-32中指令“pushl %ebp”的功能是（ ）

A

A. R[esp]←R[esp]-4，M[R[esp]]←R[ebp]

B. R[esp]←R[esp]+4，M[R[esp]]←R[ebp]

C. M[R[esp]]←R[ebp]，R[esp]←R[esp]-4

D. M[R[esp]]←R[ebp]，R[esp]←R[esp]+4

指令操作数长度与寻址方式

对于以下AT&T格式汇编指令，根据操作数的长度确定对应指令助记符中的长度后缀，并说明每个操作数的寻址方式。

- mov 8(%ebp, %ebx, 4), %ax

  后缀：w，　　源：基址+比例变址+偏移，目：寄存器

- mov %al, 12(%ebp)

  后缀：b，　　源：寄存器，目：基址+偏移

- add (, %ebx,4), %ebx

  后缀：l，　　源：比例变址，目：寄存器

- or (%ebx), %dh

  后缀：b，　　源：基址，目：寄存器

指令操作数长度与寻址方式

对于以下AT&T格式汇编指令，根据操作数的长度确定对应指令助记符中的长度后缀，并说明每个操作数的寻址方式。

- push $0xF8

  后缀：l，　　源：立即数，目：栈

- mov $0xFFF0, %eax

  后缀：l，　　源：立即数，目：寄存器

- test %cx, %cx

  后缀：w，　　源：寄存器，目：寄存器

- lea 8(%ebx, %esi), %eax

  后缀：l，　　源：基址+变址+偏移，目：寄存器

条件转移指令

根据给出采用小端方式的IA-32机器代码的反汇编结果回答下列问题：

注意：IA-32的条件转移指令都采用相对转移方式在段内直接转移，即条件转移指令的转移目标地址为：(PC) + 偏移量

（1）已知je指令的操作码为01110100，je指令的转移目标地址是什么？call指令中的转移目标地址0x80483b1是如何反汇编出来的？

804838c:	74 08	je	xxxxxxx
804838e:	e8 1e 00 00 00	call	80483b1<test>

因为je指令的操作码为01110100，所以机器代码7408H中的08H是偏移量，故转移目标地址为：0x804838c+2+0x8=0x8048396。

（2）已知jb指令的操作码为01110010，jb指令的转移目标地址是什么？movl指令中的目的地址如何反汇编出来的？

8048390:	72 f6	jb	xxxxxxx
8048392:	c6 05 00 a8 04 08 01	movl	$0x1, 0x804a800
8048399:	00 00 00		

jb指令中F6H是负偏移量（-0AH），故其转移目标地址为：0x8048390+2-0xA=0x8048388。

条件转移指令

（3）已知jle指令的操作码为01111110，mov指令的地址是什么？

xxxxxxx:	7e 16	jle	80492e0
xxxxxxx:	89 d0	mov	%edx, %eax

jle指令中的7EH为操作码，16H为偏移量，其汇编形式中的0x80492e0是转移目的地址，因此，假定后面的mov指令的地址为x，则x满足以下公式：0x80492e0=x+0x16，故x=0x80492e0-0x16=0x80492ca。

（4）已知jmp指令的转移目标地址采用相对寻址方式，jmp指令操作码为11101001，其转移目标地址是什么？

8048296:	e9 00 ff ff ff	jmp	xxxxxxx
804829b:	29 c2	sub	%eax, %edx

jmp指令中的E9H为操作码，后面4个字节为偏移量，因为是小端方式，故偏移量为FFFFFF00H，即-100H=-256。后面的sub指令的地址为0x804829b，故jmp指令的转移目标地址为0x804829b+0xffffff00=0x804829b-0x100=0x804819b。

条件转移指令

已知函数func的C语言代码框架及其过程体对应的汇编代码如下图2所示，根据对应的汇编代码填写C代码中缺失的表达式。

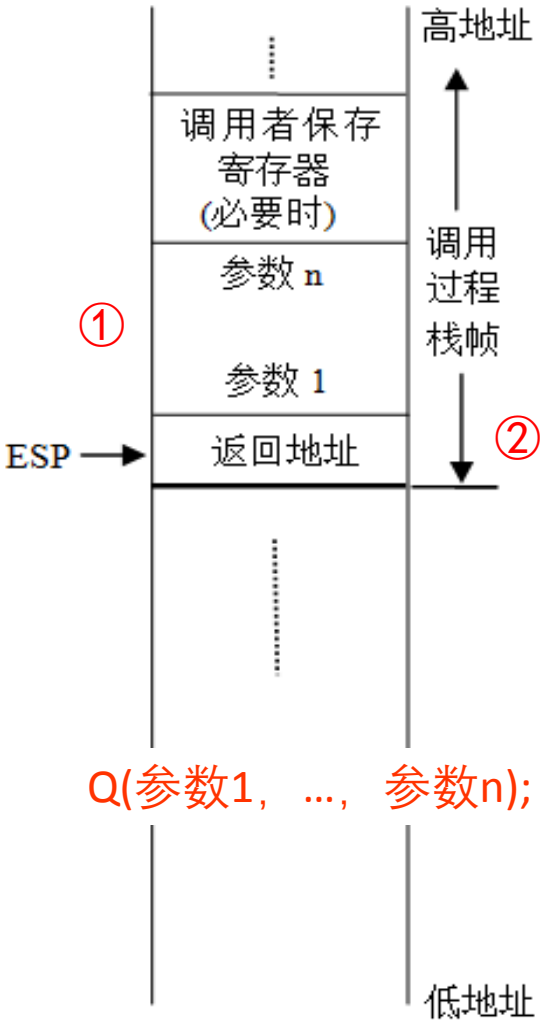
```
1   int func(int x, int y)
2   {
3       int z = ____x*y____;
4       if ( ____z<-99____ ) {
5           if  ( ___y>z___ )
6               z = ___z+y___ ;
7           else
8               z = ___z-y___ ;
9       } else if  ( ___z>=16___ )
10          z = ___z & y___ ;
11      return z;
12  }
```

```
 0:   push   %ebp
 1:   mov    %esp,%ebp
 3:   sub    $0x10,%esp
 6:   mov    0x8(%ebp),%eax
 9:   imul   0xc(%ebp),%eax
 d:   mov    %eax,-0x4(%ebp)
10:   cmpl   $0xffffff9d,-0x4(%ebp)
14:   jge    2e <func+0x2e>
16:   mov    0xc(%ebp),%eax
19:   cmp    -0x4(%ebp),%eax
1c:   jle    26 <func+0x26>
1e:   mov    0xc(%ebp),%eax
21:   add    %eax,-0x4(%ebp)
24:   jmp    3a <func+0x3a>
26:   mov    0xc(%ebp),%eax
29:   sub    %eax,-0x4(%ebp)
2c:   jmp    3a <func+0x3a>
2e:   cmpl   $0xf,-0x4(%ebp)
32:   jle    3a <func+0x3a>
34:   mov    0xc(%ebp),%eax
37:   and    %eax,-0x4(%ebp)
3a:   mov    -0x4(%ebp),%eax
3d:   leave
3e:   ret
```

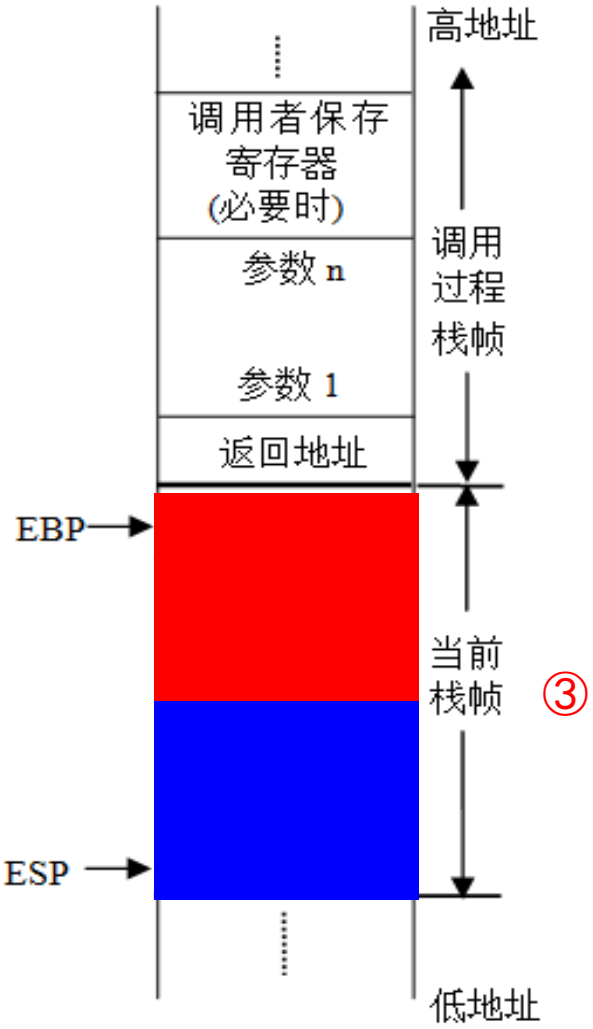
2、过程的调用

过程调用的机器级表示

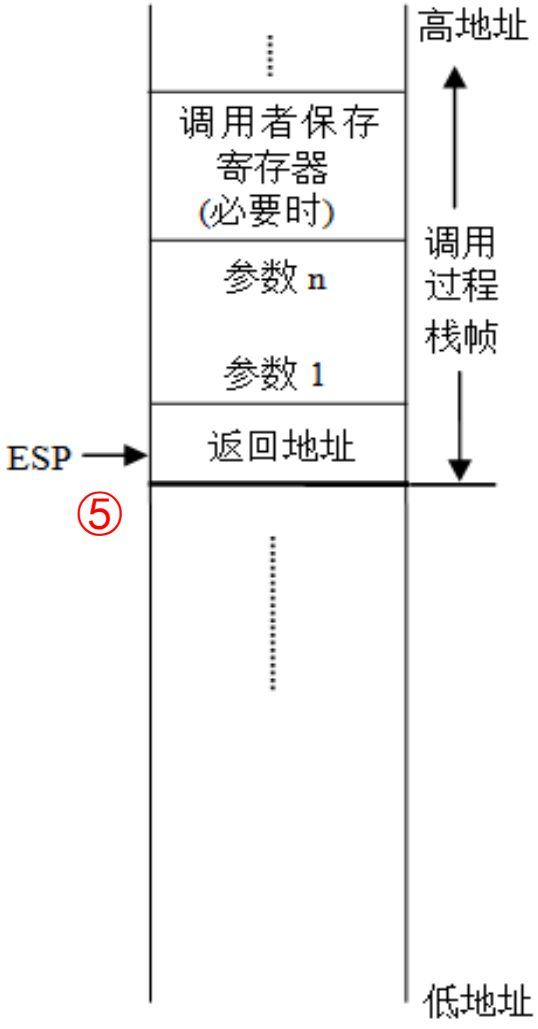
- 过程调用过程中栈和栈帧的变化 (Q为被调用过程)



(a) 过程 Q 被调用前



(b) 过程 Q 执行中



(c) 返回到过程 P 时

过程调用的机器级表示

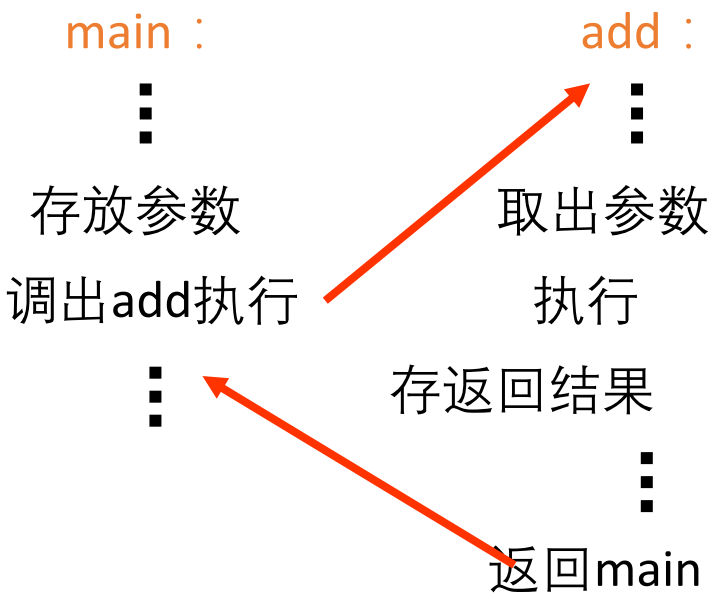
- 以下过程（函数）调用对应的机器级代码是什么？
- 如何将t1(125)、t2(80)分别传递给add中的形式参数x、y
- add函数执行的结果如何返回给caller?

```
int add ( int x, int y ) {
      return x+y;
}

int main ( ) {
      int   t1 = 125;
      int t2 = 80;
      int   sum = add (t1, t2);
      return sum;
}
```

add

main



```
int add ( int x, int y ) {
      return x+y;
}
int   caller ( ) {
      int     t1 = 125;
     int      t2 = 80;
      int     sum = add (t1, t2);
      return sum;
}
```

add

caller

caller：

```
pushl    %ebp
movl     %esp, %ebp
subl     $24, %esp
movl     $125, -12(%ebp)
movl     $80, -8(%ebp)
movl    -8(%ebp), %eax
movl     %eax, 4(%esp)
movl     -12(%ebp), %eax
movl     %eax, (%esp)
call     add
movl     %eax, -4(%ebp)
movl     -4(%ebp), %eax
leave
ret
```

准备阶段

分配局部变量

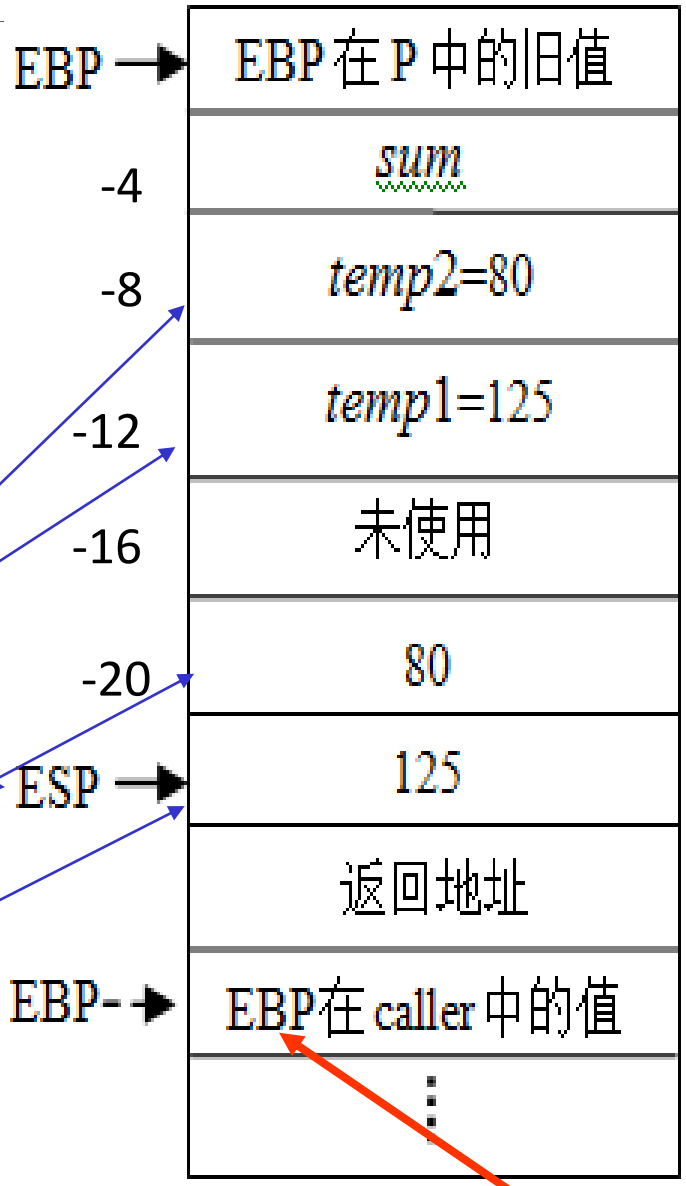
准备入口参数

返回参数总在EAX中

准备返回参数

结束阶段

```
movl   %ebp, %esp
popl   %ebp
```

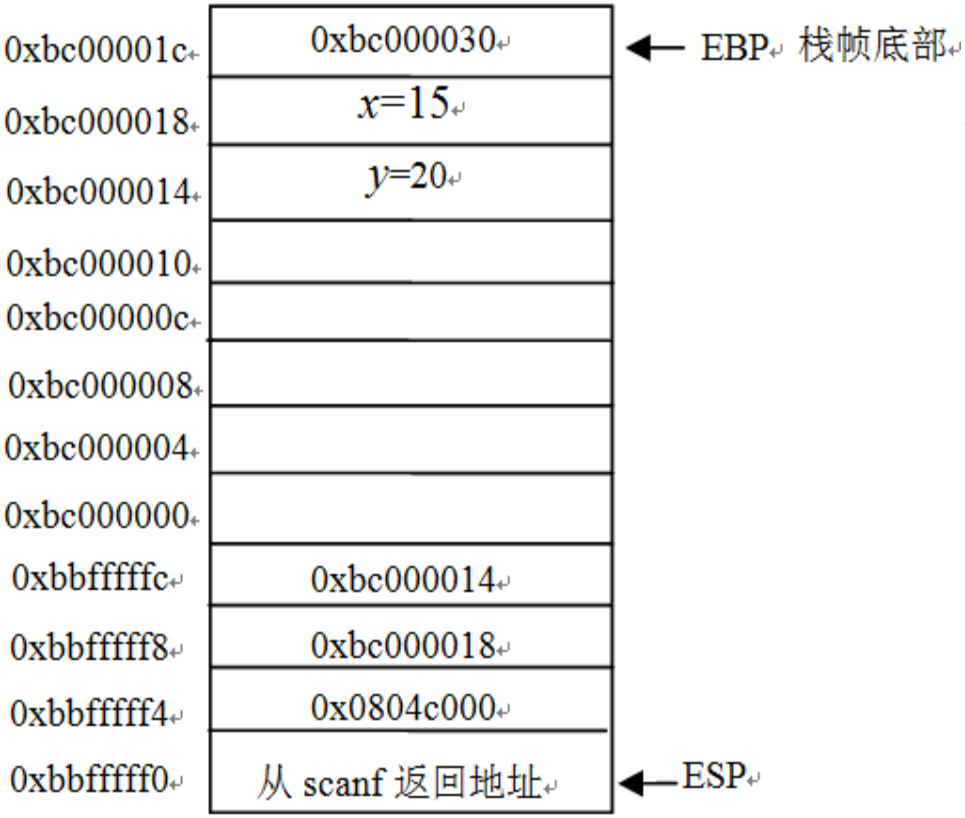


add函数开始是什么？

```
pushl   %ebp
movl   %esp, %ebp
```

过程调用

已知函数func的C语言代码如下：




```
1  func：
2    pushl  %ebp
3    movl   %esp, %ebp
4    subl                $40, %esp
5    leal                -8(%ebp), %eax
6    movl   %eax, 8(%esp)
7    leal                -4(%ebp), %eax
8    movl   %eax, 4(%esp)
9    movl   $.LC0, (%esp)
     //将指向字符串“%x %x”的指针入栈
10   call                scanf
     //假定scanf执行后x=15,y=20
11   movl   -4(%ebp), %eax
12   subl                -8(%ebp), %eax
13   leave
14   ret
```

a) 局部变量x和y所在存储单元的地址分别是什么？

&x = R[ebp]-4 = 0xbc000018

&y = R[ebp]-8 = 0xbc000014

b) 画出执行第10行指令后funct的栈帧，指出栈帧中的内容及其地址。

思考题1

已知递归函数refunc的C语言代码框架和汇编代码如下：

```
int rfun(unsigned x) {
    if ( x ==0 )
        return 0;
    unsigned nx = x >>1;
    int rv = rfun(nx);
    return x & 0x1 + rv;
}
```

```
movl    8(%ebp), %ebx
movl    $0, %eax
testl   %ebx, %eax
je      .L3
movl    %ebx, %eax
shrl    %eax
movl    %eax, (%esp)
call    rfun
movl    %ebx, %edx
andl    $1, %edx
leal    (%edx,%eax), %eax
.L3:
```

填写C代码中缺失部分，并说明函数的功能。

该rfun对应的汇编代码要能正常工作还缺少一对重要指令，是什么？

```
pushl   %ebx
....
popl    %ebx
```