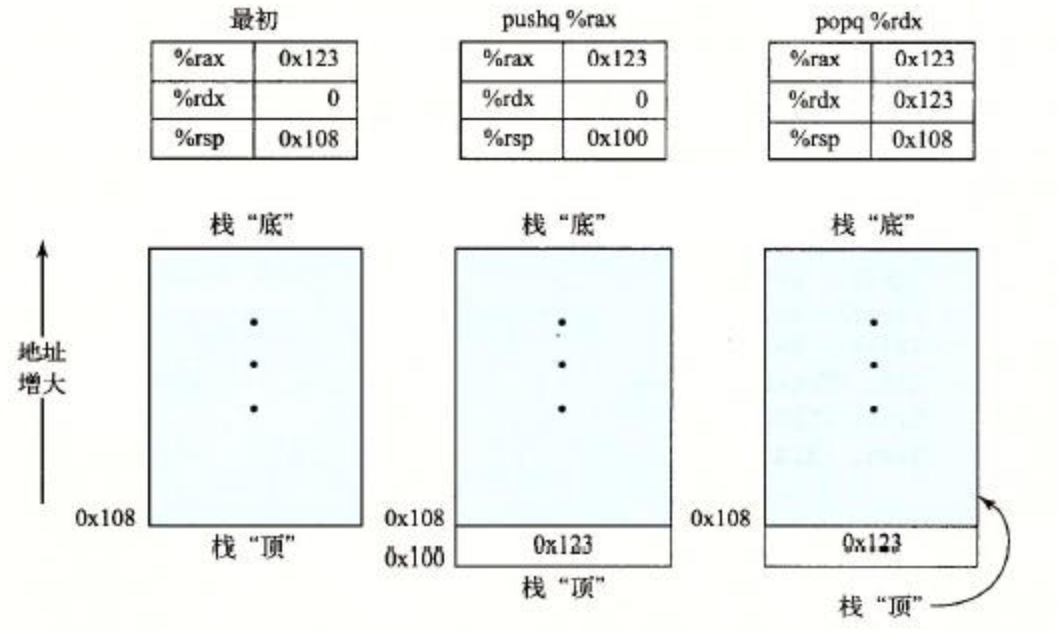
10.12机器级编程

李浩雨 施朱鸣

栈操作指令

- 栈从高地址向低地址增长
- %rsp是实指，指向实际在栈顶的内容的地址，而不是下一个空位



算术和逻辑操作：熟悉这些指令

- leaq的目的操作数是寄存器
- 注意加减法的顺序：谁减谁？结果放在哪里？
- 移位操作的移位量是立即数或者放在单字节寄存器%cl里面，移位量先mod log(数据长度)

指令		效果	描述
leaq	S, D	$D \leftarrow \&S$	加载有效地址
INC	D	$D \leftarrow D+1$	加1
DEC	D	$D \leftarrow D-1$	减1
NEG	D	$D \leftarrow -D$	取负
NOT	D	$D \leftarrow \sim D$	取补
ADD	S, D	$D \leftarrow D+S$	加
SUB	S, D	$D \leftarrow D-S$	减
IMUL	S, D	$D \leftarrow D*S$	乘
XOR	S, D	$D \leftarrow D \hat{} S$	异或
OR	S, D	$D \leftarrow D \mid S$	或
AND	S, D	$D \leftarrow D \& S$	与
SAL	k, D	$D \leftarrow D << k$	左移
SHL	k, D	$D \leftarrow D << k$	左移（等同于SAL）
SAR	k, D	$D \leftarrow D >>_A k$	算术右移
SHR	k, D	$D \leftarrow D >>_L k$	逻辑右移

算术和逻辑操作：熟悉这些指令

- 乘法用两个movq指令储存乘积

指令	效果	描述
imulq S	R[%rdx]：R[%rax]←S×R[%rax]	有符号全乘法
mulq S	R[%rdx]：R[%rax]←S×R[%rax]	无符号全乘法
clto	R[%rdx]：R[%rax]←符号扩展(R[%rax])	转换为八字
idivq S	R[%rdx]←R[%rdx]：R[%rax] mod S R[%rdx]←R[%rdx]：R[%rax]÷S	有符号除法
divq S	R[%rdx]←R[%rdx]：R[%rax] mod S R[%rdx]←R[%rdx]：R[%rax]÷S	无符号除法

图 3-12 特殊的算术操作。这些操作提供了有符号和无符号数的全 128 位乘法和除法。一对寄存器%rdx 和%rax 组成一个 128 位的八字

如何控制？条件码

- CF：发生进位（无符号溢出）
- ZF：得到零
- SF：得到负数
- OF：补码溢出
- cmp 注意比较顺序
- 常用：testq %rax %rax确定正负或者为零
- testq 0x1f %rax 检查%rax的低5个bit

指令		基于	描述
CMP	S_1，S_2	S_2-S_1	比较
cmpb			比较字节
cmpw			比较字
cmpl			比较双字
cmpq			比较四字
TEST	S_1，S_2	$S_1 \& S_2$	测试
testb			测试字节
testw			测试字
testl			测试双字
testq			测试四字

图 3-13　比较和测试指令。这些指令不修改任何寄存器的值，只设置条件码

SET指令

指令	同义名	效果	设置条件
sete *D*	setz	*D* ← ZF	相等/零
setne *D*	setnz	*D* ← ~ZF	不等/非零
sets *D*		*D* ← SF	负数
setns *D*		*D* ← ~SF	非负数
setg *D*	setnle	*D* ← ~(SF ^ OF) & ~ZF	大于（有符号>）
setge *D*	setnl	*D* ← ~(SF ^ OF)	大于等于（有符号>=）
setl *D*	setnge	*D* ← SF ^ OF	小于（有符号<）
setle *D*	setng	*D* ← (SF ^ OF) | ZF	小于等于（有符号<=）
seta *D*	setnbe	*D* ← ~CF & ~ZF	超过（无符号>）
setae *D*	setnb	*D* ← ~CF	超过或相等（无符号>=）
setb *D*	setnae	*D* ← CF	低于（无符号<）
setbe *D*	setna	*D* ← CF | ZF	低于或相等（无符号<=）

3-14 SET 指令。每条指令根据条件码的某种组合，将一个字节设置为 0 或者 1。有些指令有“同义名”，也就是同一条机器指令有别的名字

跳转和条件跳转

- 直接跳转：跳到一个标号 jmp .L1
- 间接跳转：跳到寄存器或内存中读到的地址 jmp *%rax 到寄存器%rax中的地址，jmp *(%rax) 到%rax存的地址对应的内存位置找到一个数，作为跳转目标的地址
- PC相对寻址：目标指令-下一条指令
- 绝对寻址：4bytes地址

指令	同义名	跳转条件	描述
jmp *Label*		1	直接跳转
jmp **Operand*		1	间接跳转
je *Label*	jz	ZF	相等/零
jne *Label*	jnz	~ZF	不相等/非零
js *Label*		SF	负数
jns *Label*		~SF	非负数
jg *Label*	jnle	~(SF ^ OF) & ~ZF	大于（有符号>）
jge *Label*	jnl	~(SF ^ OF)	大于或等于（有符号>=）
jl *Label*	jnge	SF ^ OF	小于（有符号<）
jle *Label*	jng	(SF ^ OF) | ZF	小于或等于（有符号<=）
ja *Label*	jnbe	~CF & ~ZF	超过（无符号>）
jae *Label*	jnb	~CF	超过或相等（无符号>=）
jb *Label*	jnae	CF	低于（无符号<）
jbe *Label*	jna	CF | ZF	低于或相等（无符号<=）

3-15 jump 指令。当跳转条件满足时，这些指令会跳转到一条带标号的目的地。有些指令有“同义名”，也就是同一条机器指令的别名

PC寻址：目标地址-下一指令地址

```
1     movq    %rdi, %rax
2     jmp     .L2
3   .L3:
4     sarq    %rax
5   .L2:
6     testq   %rax, %rax
7     jg      .L3
8     rep; ret
```

汇编器产生的“.o”格式的反汇编版本如下：

```
1     0:   48 89 f8          mov    %rdi,%rax
2     3:   eb 03             jmp    8 <loop+0x8>
3     5:   48 d1 f8          sar    %rax
4     8:   48 85 c0          test   %rax,%rax
5     b:   7f f8             jg     5 <loop+0x5>
6     d:   f3 c3             repz retq
```

下面是链接后的程序反汇编版本：

```
1     4004d0:   48 89 f8          mov    %rdi,%rax
2     4004d3:   eb 03             jmp    4004d8 <loop+0x8>
3     4004d5:   48 d1 f8          sar    %rax
4     4004d8:   48 85 c0          test   %rax,%rax
5     4004db:   7f f8             jg     4004d5 <loop+0x5>
6     4004dd:   f3 c3             repz retq
```

条件分支长什么样子？

```
long lt_cnt = 0;
long ge_cnt = 0;

long absdiff_se(long x, long y)
{
    long result;
    if (x < y) {
        lt_cnt++;
        result = y - x;
    }
    else {
        ge_cnt++;
        result = x - y;
    }
    return result;
}
```

```
    long absdiff_se(long x, long y)
    x in %rdi, y in %rsi
1   absdiff_se:
2     cmpq    %rsi, %rdi          Compare x:y
3     jge     .L2                 If >= goto x_ge_y
4     addq    $1, lt_cnt(%rip)    lt_cnt++
5     movq    %rsi, %rax
6     subq    %rdi, %rax          result = y - x
7     ret                         Return
8   .L2:                        x_ge_y:
9     addq    $1, ge_cnt(%rip)    ge_cnt++
10    movq    %rdi, %rax
11    subq    %rsi, %rax          result = x - y
12    ret                         Return
```

条件分支的替代：条件传送

- v = x ? y : z
- 不允许有副作用。不允许在运算y和z时修改了条件

指令	同义名	传送条件	描述
cmove S, R	cmovz	ZF	相等/零
cmovne S, R	cmovnz	~ZF	不相等/非零
cmovs S, R		SF	负数
cmovns S, R		~SF	非负数
cmovg S, R	cmovnle	~(SF ^ OF) & ~ZF	大于（有符号>）
cmovge S, R	cmovnl	~(SF ^ OF)	大于或等于（有符号>=）
cmovl S, R	cmovnge	SF ^ OF	小于（有符号<）
cmovle S, R	cmovng	(SF ^ OF) | ZF	小于或等于（有符号<=）
cmova S, R	cmovnbe	~CF & ~ZF	超过（无符号>）
cmovae S, R	cmovnb	~CF	超过或相等（无符号>=）
cmovb S, R	cmovnae	CF	低于（无符号<）
cmovbe S, R	cmovna	CF | ZF	低于或相等（无符号<=）

do-while循环：字面意思

```
do
    body-statement
    while (test-expr);
```

```
loop:
    body-statement
    t = test-expr;
    if (t)
        goto loop;
```

```
long fact_do(long n)
{
    long result = 1;
    do {
        result *= n;
        n = n-1;
    } while (n > 1);
    return result;
}
```

a）C代码

```
long fact_do_goto(long n)
{
    long result = 1;
 loop:
    result *= n;
    n = n-1;
    if (n > 1)
        goto loop;
    return result;
}
```

b）等价的goto版本

```
    long fact_do(long n)
    n in %rdi
1   fact_do:
2     movl    $1, %eax        Set result = 1
3   .L2:                    loop:
4     imulq   %rdi, %rax      Compute result *= n
5     subq    $1, %rdi        Decrement n
6     cmpq    $1, %rdi        Compare n:1
7     jg      .L2             If >, goto loop
8     rep; ret                Return
```

c）对应的汇编代码

while循环-O1 jump to middle

```
while (test-expr)
    body-statement
```

```
    goto test;
loop:
    body-statement
test:
    t = test-expr;
    if (t)
        goto loop;
```

```
long fact_while(long n)
{
    long result = 1;
    while (n > 1) {
        result *= n;
        n = n-1;
    }
    return result;
}
```

a）C代码

```
long fact_while_jm_goto(long n)
{
    long result = 1;
    goto test;
 loop:
    result *= n;
    n = n-1;
 test:
    if (n > 1)
        goto loop;
    return result;
}
```

b）等价的goto版本

```
  long fact_while(long n)
  n in %rdi
fact_while:
  movl    $1, %eax        Set result = 1
  jmp     .L5             Goto test
.L6:                    loop:
  imulq   %rdi, %rax      Compute result *= n
  subq    $1, %rdi        Decrement n
.L5:                    test:
  cmpq    $1, %rdi        Compare n:1
  jg      .L6             If >, goto loop
  rep; ret                Return
```

c）对应的汇编代码

while循环-O2
检查，结束或do-while

```
while (test-expr)
    body-statement
```

```
t = test-expr;
if (!t)
    goto done;
loop:
    body-statement
    t = test-expr;
    if (t)
        goto loop;
done:
```

```
long fact_while(long n)
{
    long result = 1;
    while (n > 1) {
        result *= n;
        n = n-1;
    }
    return result;
}
```

a）C代码

```
long fact_while_gd_goto(long n)
{
    long result = 1;
    if (n <= 1)
        goto done;
 loop:
    result *= n;
    n = n-1;
    if (n != 1)
        goto loop;
 done:
    return result;
}
```

b）等价的goto版本

```
    long fact_while(long n)
    n in %rdi
 1  fact_while:
 2    cmpq    $1, %rdi        Compare n:1
 3    jle     .L7             If <=, goto done
 4    movl    $1, %eax        Set result = 1
 5  .L6:                    loop:
 6    imulq   %rdi, %rax      Compute result *= n
 7    subq    $1, %rdi        Decrement n
 8    cmpq    $1, %rdi        Compare n:1
 9    jne     .L6             If !=, goto loop
10    rep; ret                Return
11  .L7:                    done:
12    movl    $1, %eax        Compute result = 1
13    ret                     Return
```

c）对应的汇编代码

for循环：转两种while循环

```
for (init-expr; test-expr; update-expr)
    body-statement
```

```
init-expr;
while (test-expr) {
    body-statement
    update-expr;
}
```

switch跳转表不是不停if else

```
    void switch_eg(long x, long n, long *dest)
    x in %rdi, n in %rsi, dest in %rdx
1   switch_eg:
2     subq    $100, %rsi              Compute index = n-100
3     cmpq    $6, %rsi                Compare index:6
4     ja      .L8                     If >, goto loc_def
5     jmp     *.L4(,%rsi,8)           Goto *jt[index]
6   .L3:                            loc_A:
7     leaq    (%rdi,%rdi,2), %rax       3*x
8     leaq    (%rdi,%rax,4), %rdi       val = 13*x
9     jmp     .L2                       Goto done
10  .L5:                            loc_B:
11    addq    $10, %rdi                 x = x + 10
12  .L6:                            loc_C:
13    addq    $11, %rdi                 val = x + 11
14    jmp     .L2                       Goto done
15  .L7:                            loc_D:
16    imulq   %rdi, %rdi                val = x * x
17    jmp     .L2                       Goto done
18  .L8:                            loc_def:
19    movl    $0, %edi                  val = 0
20  .L2:                            done:
21    movq    %rdi, (%rdx)              *dest = val
22    ret                               Return
```

```
1     .section      .rodata
2     .align 8        Align address to multiple of 8
3   .L4:
4     .quad   .L3     Case 100: loc_A
5     .quad   .L8     Case 101: loc_def
6     .quad   .L5     Case 102: loc_B
7     .quad   .L6     Case 103: loc_C
8     .quad   .L7     Case 104: loc_D
9     .quad   .L8     Case 105: loc_def
0     .quad   .L7     Case 106: loc_D
```

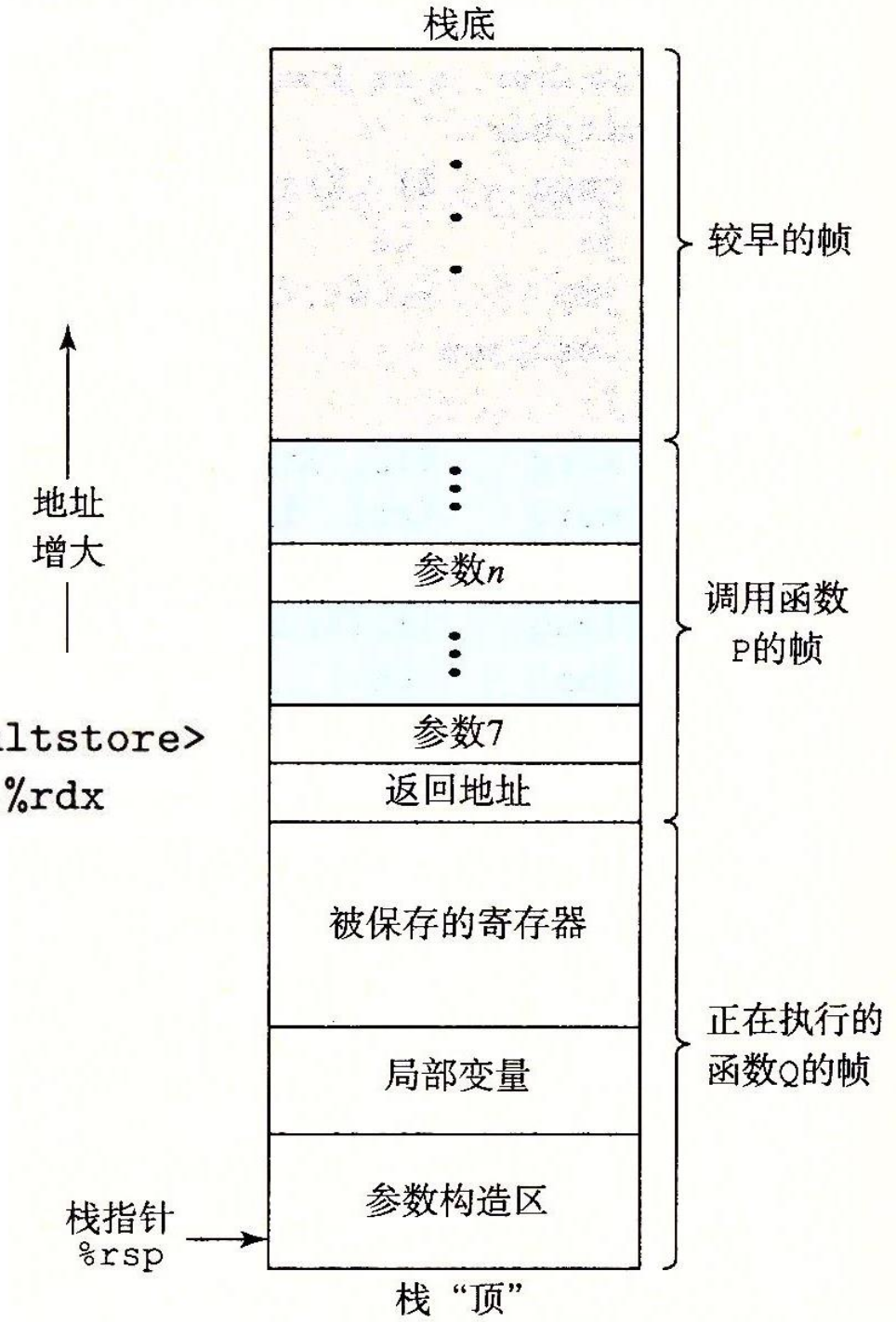
运行时栈的结构

- call指令的直接调用，注意机器码PC寻址还是绝对寻址

  call <multstore>

```
Call to multstore from main
5    400563:   e8 d8 ff ff ff          callq   400540 <multstore>
6    400568:   48 8b 54 24 08          mov     0x8(%rsp),%rdx
```

- 间接调用call *(%rax)



栈与函数调用

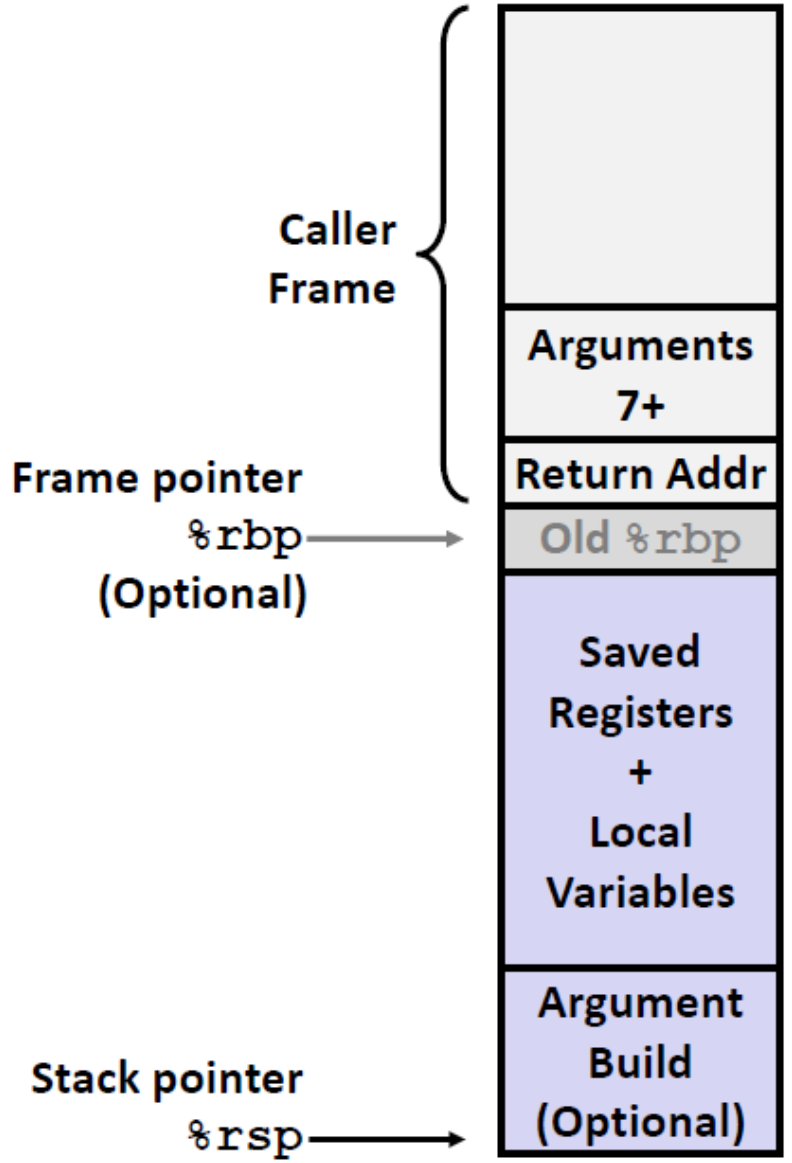
- 能力要求：快速阅读并理解汇编代码，综合性强，需要足够熟练。据往年看，期中考试中难度仅次于流水线。
- 题型：选择题；非选择题（C代码，汇编代码的对照填空，补全栈中数据）

- 概念明确：寄存器组是唯一被所有过程共享的资源
- 什么时候用到栈?
  - 寄存器不够
  - 对局部变量取址
  - 要求局部变量能够用指针访问

栈与函数调用

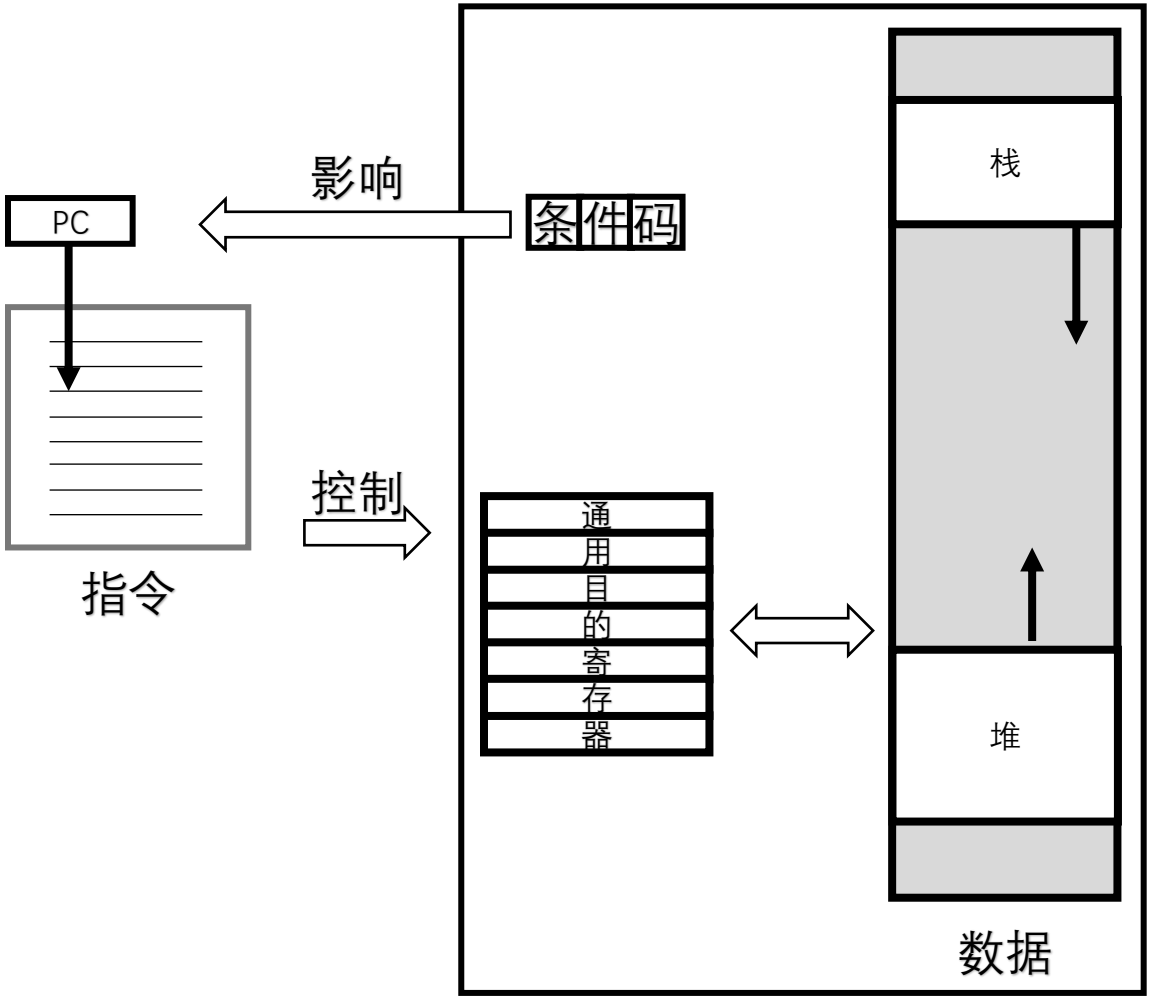
- call指令
  - 参数压栈（参数大于6个时，按照地址由低到高的顺序存储在栈中，栈上数据8字节对齐）
  - 返回地址入栈（注意是下一条指令的地址，不是当前指令的地址）
  - ……其他操作

- 栈上局部存储（64位为例）
  - 寄存器传参 %rdi %rsi %rdx %rcx %r8 %r9
  - 返回值%rax
  - 被调用者保存寄存器：%rbx %rbp %r12-%r15；调用者保存寄存器：除了%rsp的其他



栈与函数调用

- 一种不严格的理解方式（在学完Cache和链接之后会严格化）
- 一个程序的执行需要有指令和数据两部分
- 计算机内存中有一部分空间存放指令(第七章会学)，也有专门的部分存放数据(例如堆栈，但并不全在堆栈中)
- PC指向可执行的代码段。处理器从PC的位置读到指令，然后执行指令（如把内存中的数据读入寄存器，寄存器中数据写入内容，寄存器之间做操作等）。
- 指令执行过程中可能设置条件码，这些条件码可能影响PC指向的位置（指令的执行顺序不止依赖于代码，还可能依赖于所给的数据）
- 需要使用很大栈空间的程序代码未必会很长（很短的代码如果有特别多次函数调用，那么栈就会很长），需要很多数据的程序代码也未必很长（因为这些数据不存放在代码段）
- 不要把PC，条件码寄存器，通用目的寄存器搞混；也不要把代码和数据弄混



练习

```
    Disassembly of last(long u, long v)
    u in %rdi, v in %rsi
1   0000000000400540 <last>:
2     400540:   48 89 f8              mov    %rdi,%rax        L1: u
3     400543:   48 0f af c6           imul   %rsi,%rax        L2: u*v
4     400547:   c3                    retq                    L3: Return

    Disassembly of first(long x)
    x in %rdi
5   0000000000400548 <first>:
6     400548:   48 8d 77 01           lea    0x1(%rdi),%rsi   F1: x+1
7     40054c:   48 83 ef 01           sub    $0x1,%rdi        F2: x-1
8     400550:   e8 eb ff ff ff        callq  400540 <last>    F3: Call last(x-1,x+1)
```

指令			状态值(指令执行前)					
标号	PC	指令	%rdi	%rsi	%rax	%rsp	* %rsp	描述
M1	0x400560	callq	10	—	—	0x7fffffffe820	—	调用 first(10)
F1	0x400548	lea	10	--	--	0x7fffffffe818	0x400565	first的入口
F2								
F3								
L1								
L2								
L3								
F4								
M2								

*图片和动画来自往年孙雨奇同学的回课ppt