三.程序的机器级表示

1.历史观点

2.程序编码

3.数据格式

4.访问信息

4.1操作数指示符

4.2数据传输指令

4.3数据传输实例

复习一下

实例：最简单形式的数据传输指令---MOV类

指令：4条

movb

movw

movl

move

作用：将数据从一个位置复制到另一个位置

区别：操作的数据大小不同， 分别是1， 2， 4， 8 个字节IA32指令集

（Intel 32 位体系结构Intel Architecture 32-bit）熟称x86

看书上的代码

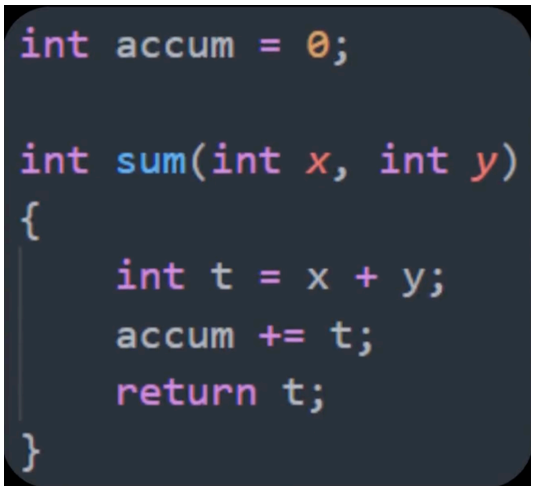
4.4压入和弹出栈数据

- 因算数和逻辑操作在汇编层面都是都内存，寄存器，和立即数的操作，（还有一个栈的概念）

所以我们要先理解三者的位置和关系

书上第6，7页也有图

这里的汇编基于



如何通俗地解释什么是寄存器?

匿名用户

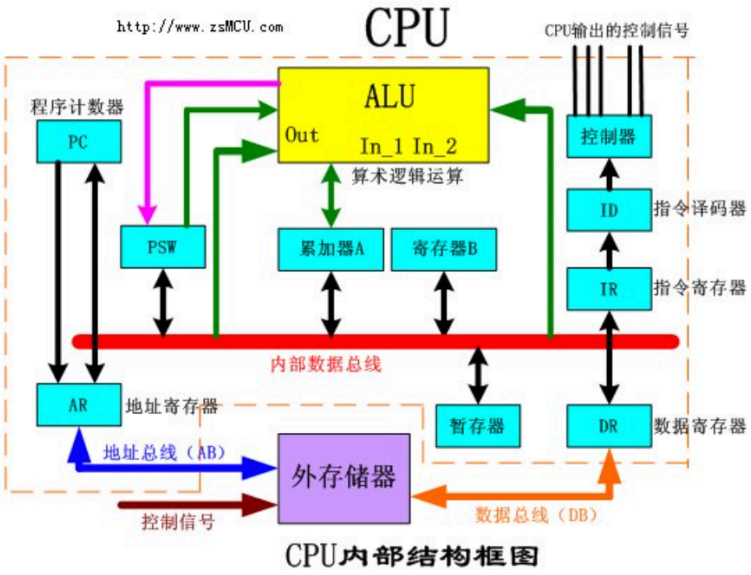
寄存器就是你的口袋。身上只有那么几个，只装最常用或者马上要用的东西。
内存就是你的背包。有时候拿点什么放到口袋里，有时候从口袋里拿出点东西放在背包里。
辅存就是你家里的抽屉。可以放很多东西，但存取不方便。

发布于 2014-08-08

goole度娘
推荐于2017-09-19 · TA获得超过7062个赞

关注

1. 寄存器（Cache）是CPU内部集成的,内存是挂在CPU外面的数据总线上的,访问内存时要在CPU的寄存器（Cache）填上地址,再执行相应的汇编指令,这时CPU会在数据总线上生成读取或写入内存数据的时钟信号,最终内存的内容会被CPU寄存器（Cache）的内容更新(写入)或者被读入CPU的寄存器（Cache）(读取)。如图：



CPU内部结构框图

2.延伸阅读：CPU、内存、寄存器之间的关系cpu 取址 –>地址输入地址寄存器 –> 缓存命中即，则数据进入数据寄存器 –> 缓存未命中则进入内存 –> 内存TLB快表命中则数据块进入缓存，数据进入寄存器 –> 内存TLB快表未命中则局部数据块进入缓存和快表 –> 内存未命中则进入硬盘虚拟存储区

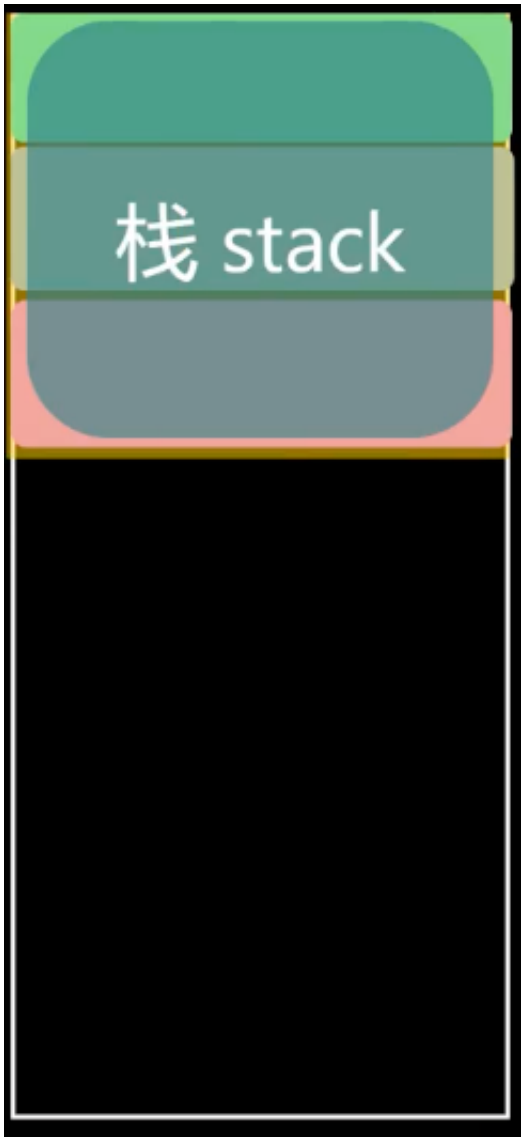
栈是什么?

栈是内存中属于某个函数的一段连续的空间

一个程序在运行时，这些局部变量会会存储在内存中的区域，称为 栈

```
int accum = 0;

int sum(int x, int y)
{
    int t = x + y;
    accum += t;
    return t;
}
```



栈是一种基本的数据结构，只有两个操作，一个是push入栈，另一个是pop出栈

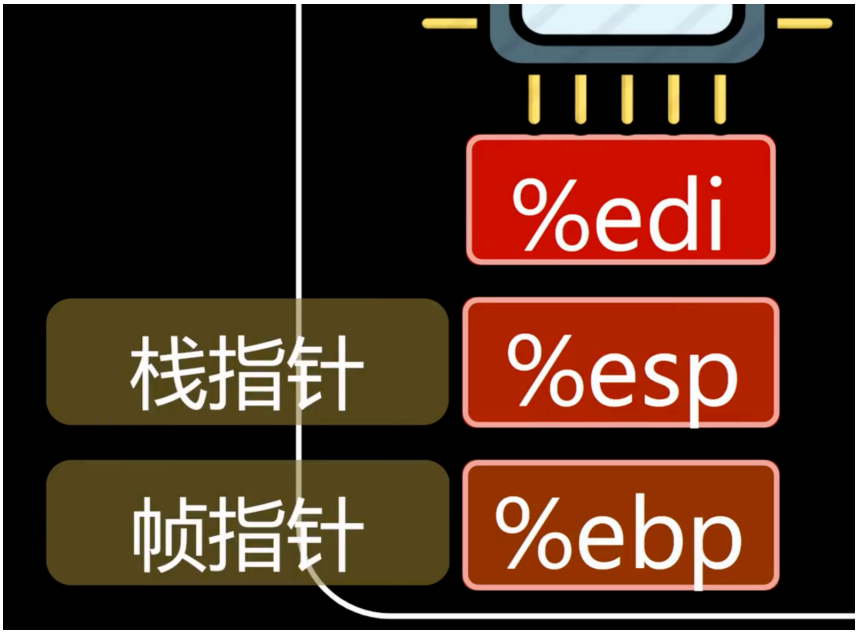
pushq是将数据压到栈上

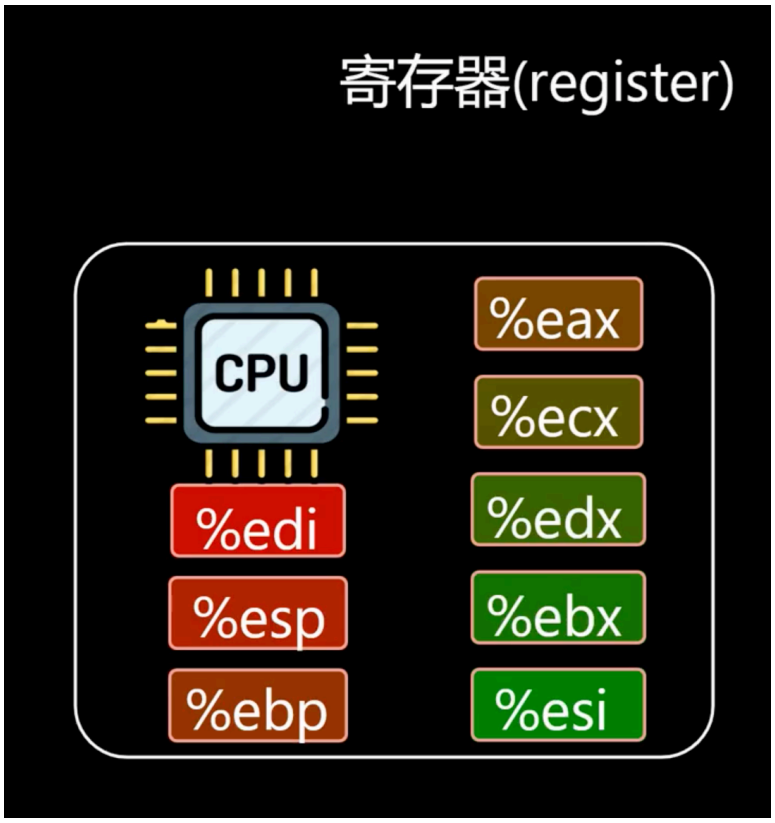
popq是将弹出 数据

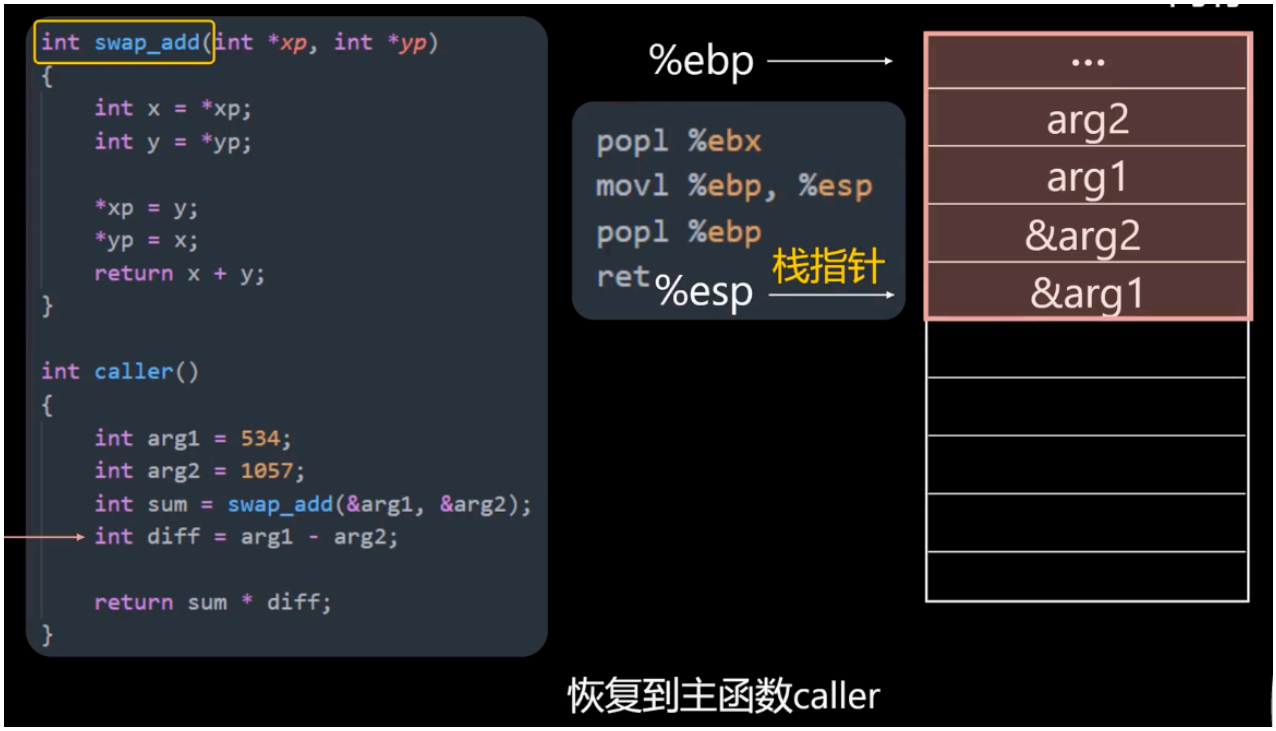
但在内存中，栈是倒过来的，也就是从高地址向低地址增长

这样栈顶的指针称为栈指针，指向栈的最下面，这里用寄存器%esp（stack pointer）

解释书上的代码


%edi
栈指针
%esp
帧指针
%ebp


寄存器(register)
CPU
%eax
%ecx
%edx
%ebx
%esi
%edi
%esp
%ebp



帧指针用来标志一段程序的开始

一个函数运行前会先创建好栈帧，主函数对应的栈帧的起始地址用帧指针%ebp保存

栈指针和帧指针操作函数（程序）的进行

5.算数和逻辑操作

对数据的指令

立即数寻址、寄存器寻址、直接寻址、间接寻址

王建勇，网络工程从业者

寻址就是如何找到程序想要使用的数据，根据不同的途径分为立即数寻址、直接寻址、寄存器寻址、间接寻址等多种寻址方式。

立即数寻址：

严格来说，立即数寻址不应该称为一种寻址方式，因为程序想要使用的数据已经摆在那里了，不需要寻址，直接就可以使用。如：

mov eax, 0x1; //将数值1赋值给eax寄存器。

寄存器寻址：

程序想要访问的数据存储在寄存器中，直接访问寄存器就可以获取到，这种方式称为寄存器寻址。如

mov ebx，eax; //将eax寄存器中的值赋值给ebx寄存器。

直接寻址：

程序想要访问的数据是直接给出，就是在寄存器或者内存单元中，不会在其他地方了。

直接寻址：

程序想要访问的数据是直接给出，就是在寄存器或者内存单元中，不会在其他地方了。

那么程序想要访问内存，就需要明确访问哪个内存单元的数据？这种通过地址直接访问内存单元中数据的方式就是直接寻址。如：

mov ebx,[0x00401000]; //将内存单元0x00401000中的数据赋值给ebx寄存器。严格来说是将该地址开始的连续4个byte值赋值给ebx寄存器。

间接寻址：

程序想要访问的数据存储在内存单元中，但是该单元的地址程序目前不知道是多少，需要通过运算等方式动态获取，然后这种间接寻址到内存单元的方式称为间接寻址：

mov eax, 0x00401000;

mov ebx,[eax];

mov ecx,[eax + 1];

发布于 02-04

操作：一共有四组

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

5.1加载有效地址

leap指令：它的指令形式是从内存读取数据到寄存器，但实际上它根本没有引用内存。它的第一个操作数看上去是一个内存引用，但该指令并不是从指定的位置读入数据，而是将有效地址写入到目的操作数。

源操作数 目的操作数

内存	|	必须是一个寄存器

5.2一元和二元操作

一元操作：一个操作数，既是源操作数，又是目的操作数。可以是一个寄存器，也可以是一个内存位置。

C语言：加1运算符（++）和 减1运算符（——）

二元操作：第二个操作数既是 源 又是 目的 操作数，

C语言：赋值运算符 （x += y）

源操作数 目的操作数

立即数	寄存器
寄存器	内存位置
内存位置	

5.3移位操作

移位操作：第一项给出移位量，第二项给出要移位的数。移位量可以是一个立即数，也可以放在单字节寄存器 %cl 中

C语言：移位运算符（>>和<<）

源操作数	目的操作数
立即数	寄存器
寄存器	内存位置

5.4讨论

无符号和有符号？？？

书上给了一段代码

5.5特殊的算数操作 （128位乘积以及整数除法）

imulq指令有两种不同的形式（）双胞胎

1.图3-10中IMUL指令类中一中双操作数

2. 2条单操作数乘法指令

    1）无符号数乘法（mulq）

2）补码乘法（imulq）

~~有符号无符号好烦~~

除法

```
被除数   /    除数     =    商   。。。。 余数

被除数：idivl指令从%rdx中取出
除数：作为指令的操作数给出
商：放到%rax中
余数：放在%rdx中
```

6.控制

7.过程

8.数组分配和访问

9.异质的数据结构

10.在机器级程序中将程序和控制结合起来

11.浮点代码

12.小结