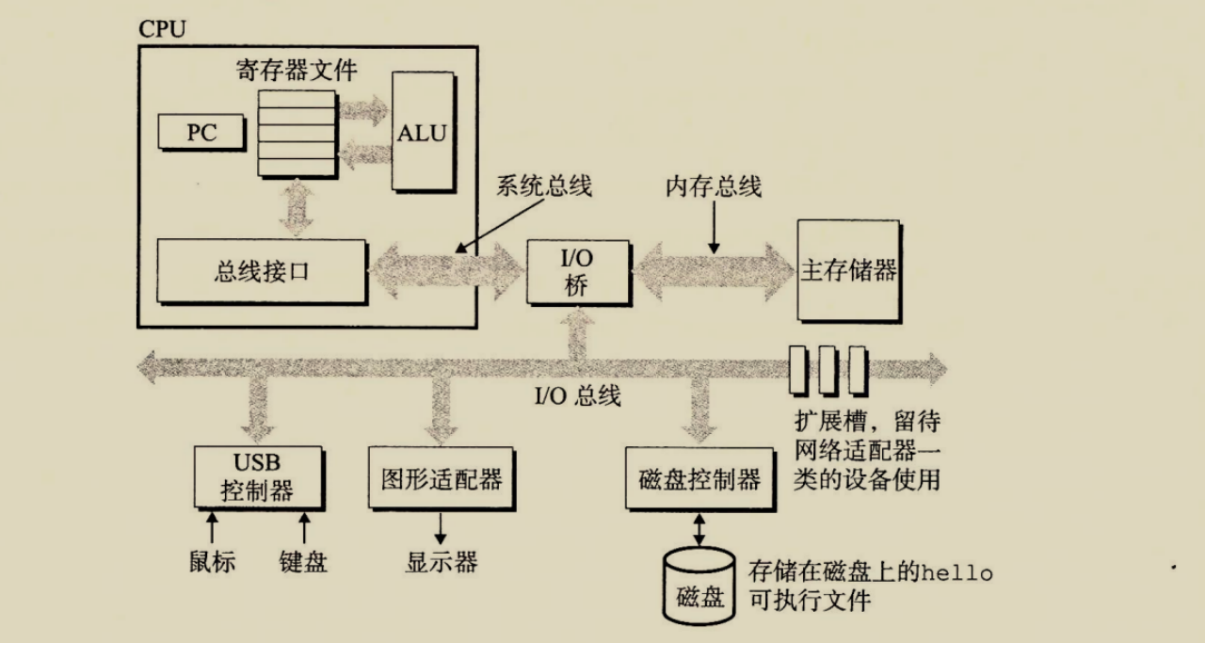
ICS第一次回课

曾为帅 20230927

从计算机系统的硬件说起

如果想要了解对应的机器语言，我们必须知道其是用什么结构说话的



- 基本组成部分
  - CPU
  - 主存储器（内存）
  - I/O外部设备
  - I/O总线

- I/O总线
  - 实际上是贯穿整个系统的一组电子管道，作用就是携带信息在各个部件之间传递
  - 总线被设计为定长的字节块——字
  - 一个字中的字节数是一个基本的系统参数
  - 32位机器——字长为4（4个字节），64位机器——字长为8（8个字节）

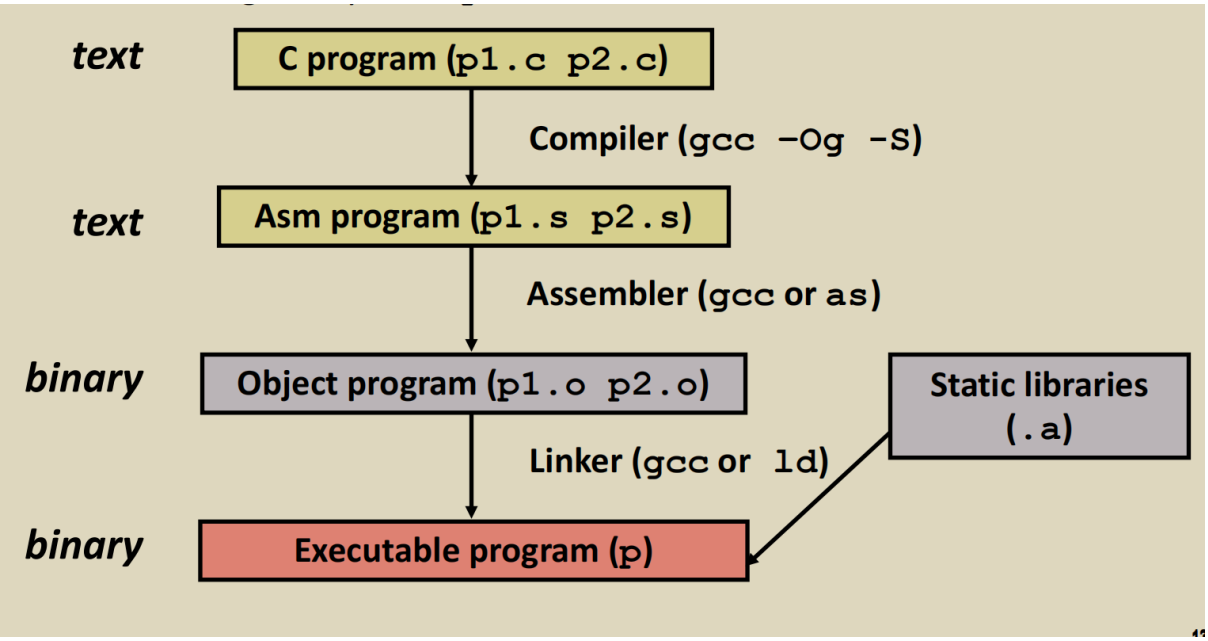
- CPU：中央处理器
  - Intel Processors and Architectures
    - Architecture
      - 什么是体系结构
        - The parts of a processor design that one needs to understand or write assembly/machine code
        - 计算机分为对应的硬件基础和相应实现的软件，所以我们需要提前规定好对应软件和硬件的接口，使得软件和硬件可以分别发展最后只需要把它们两耦合起来即可（就像老师上课提到的 自行车和表演 的例子）
        - 体系结构就是对于这样一套接口的规定
        - 通俗的讲说也就是32位的机器对应就要和32位的系统相对应，64位的机器就要和64位的系统相对应
      - IA32:Intel32位体系结构
      - x86-64:IA32的64位扩展
      - 向后兼容性：旧的软件可以在新的体系结构上运行，新的软件不能在旧的体系结构上运行
    - Processor
      - Intel处理器的主要发展过程：16位——32位——64位（结构）——多核（数量）
      - 多核处理器指的是将多个处理器集成到一个芯片上，每一个对应的处理器就称为一个核
      - 处理器组成：

    - 程序计数器PC：一个大小为一个字的存储设备（记为%rip)，指向对应主存中的下一条指令在内存中的地址（留着，等会讲到内存的时候说）
    - 寄存器文件：一个小的存储设备，由一些单个字长的寄存器组成
        - 一些——现在是16个用来存储64位的值的寄存器（整数寄存器），但是之前有变化，和对应的条件码寄存器
        - 单个字长——现在是64位，但是此前也有32位，16位的
    - ALU；算术与逻辑单元——计算新的数据核地址值
        - ALU不是条件码寄存器

- 主存
    - 逻辑上，存储器就是一个线性的字节数组的，每个字节有唯一的地址，对应的地址从0开始。
    - 结合上面对于PC的考量，我们考虑对应的这样一个字节数组最多为多少
        - 以32位为例，PC可以表示2^32个值，相应的也能对应这么多个字节、这么多数据

现在我们已经知道了对应的机器是基于什么样的硬件结构，下面我们则要具体考虑我们所说的机器语言

再看一个程序的生命进程



- 编译器：将原始的C代码转化为对应的汇编代码
- 汇编器（反汇编器）：将对应的汇编代码转化成二进制目标代码
  - 目标代码是机器代码的一种形式，包含所有指令的二进制表示，但是还没有指定对应的全局地址，这个工作就是对应链接器做的（找位置）
- 链接器：将目标代码文件与实现库函数的代码合并，产生最终的可执行代码文件
  - 实现了不同文件之间的引用

再就到了我们的汇编语言

数据格式

C声明	Intel数据类型	汇编代码后缀	大小(字节)
char	字节	b	1
short	字	w	2
int	双字	l	4
long	四字	q	8
char*	四字	q	8
float	单精度	s	4
double	双精度	l	8

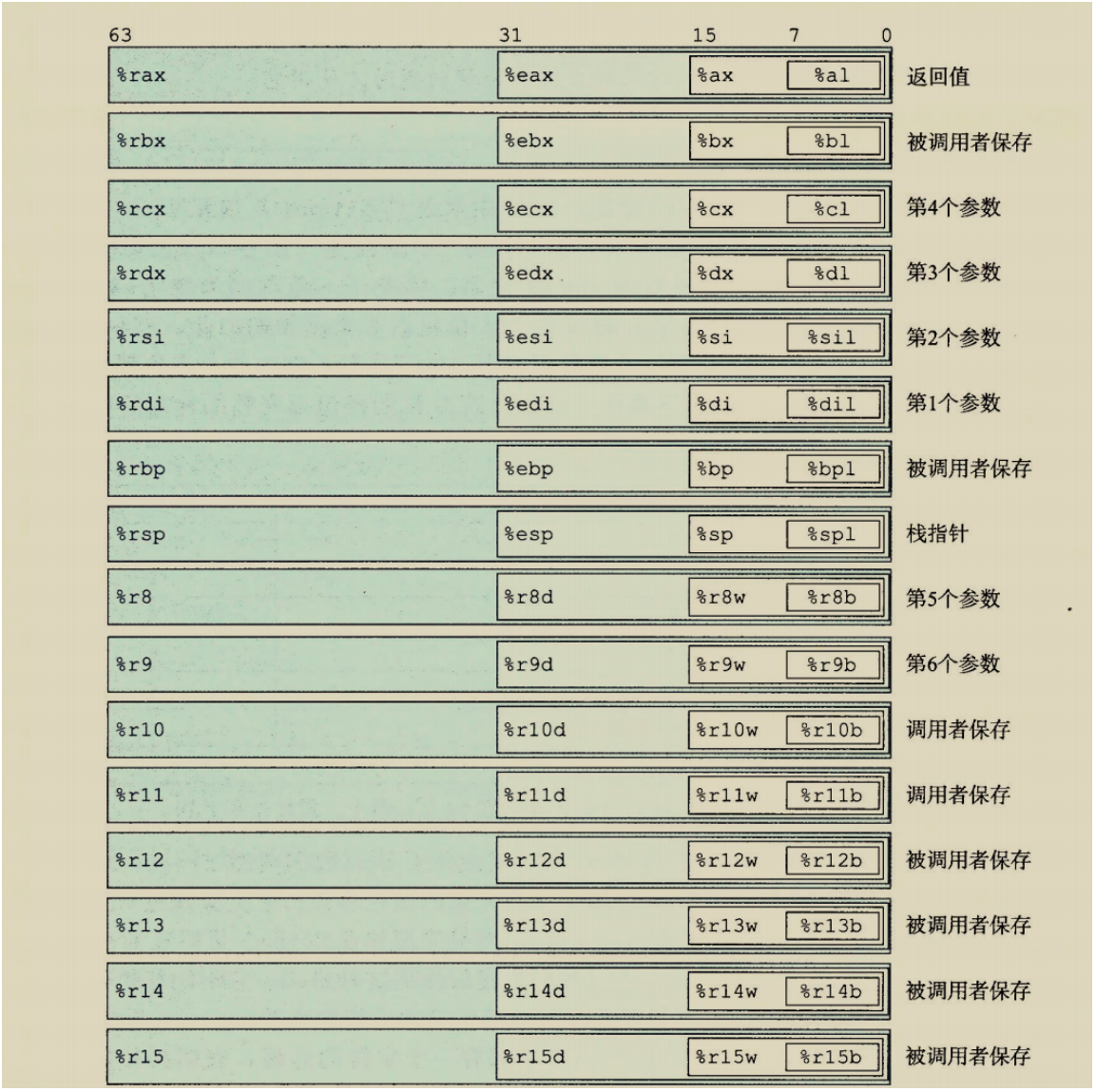
Intel用字表示16位数据类型

其中char*表示的是一个指针，这是machine-specific的

GCC编译器生成的汇编代码指令都有一个字符的后缀表明对应操作数的大小

访问信息

- 寄存器的不同层次（信息存储在哪里）



不同位的操作可以访问最低的不同位数的字节

特别要注意的是：生成1字节和2字节数字的指令会保持剩下的字节不变；生成4字节数字的指令会把高位4个字节置放为0（所以有些时候是很有用的）

且这些寄存器都称为整数寄存器，但实际上还有条件码寄存器等

- 操作数指示符（进行操作的对象）
  - 三种操作数
    - 立即数：用$跟上一个常数
    - 寄存器：访问某个寄存器的内容
    - 内存：根据计算出来的地址访问某个内存位置
  - 一般操作的表示

$$Oper \quad Src, Des$$

  - 对应的寻址方法（Address mode expression）

$$Imm(r_b, r_i, s) = Imm + R[r_b] + R[r_i] \times s$$

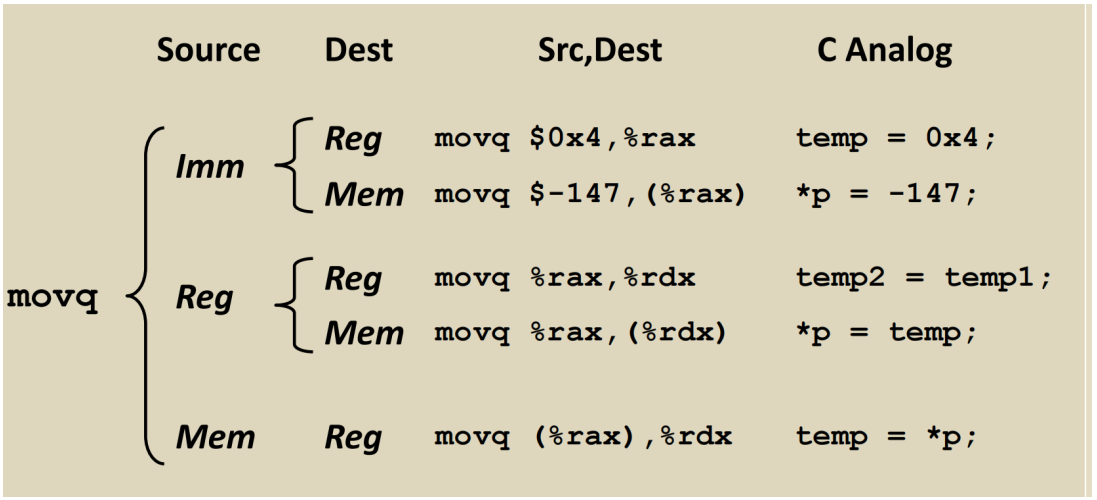
特别注意的是对应的rb 省略的时候也要打逗号

而且对应的s应该是1、2、4、8，保证对应数据的完整性

- 数据传送指令
  - 传送指令的的基本形式

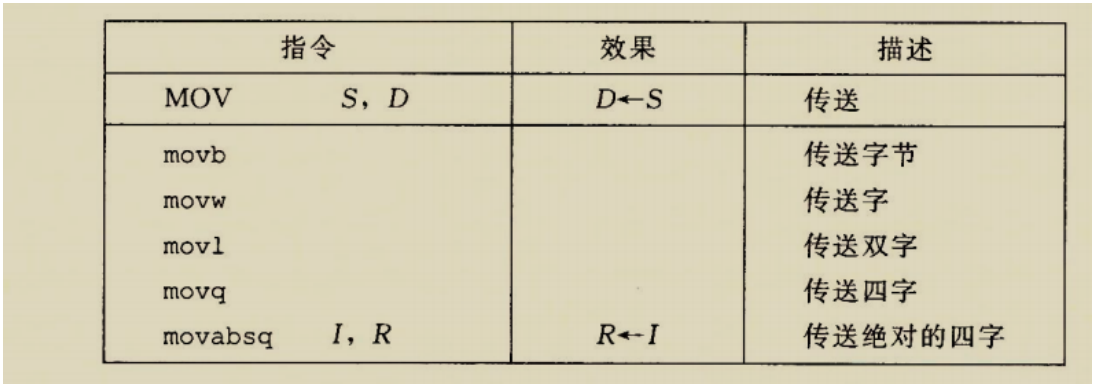
$$movq \quad Src \quad Dest$$

  - 不同的数据传输方式



- 切记要注意对应的移动操作不能进行内存到内存的直接移动，必须先将对应的数据加载到寄存器中作为中转让后再转移

- 传输不同类型的数据



指令		效果	描述
MOV	S，D	D←S	传送
movb			传送字节
movw			传送字
movl			传送双字
movq			传送四字
movabsq	I，R	R←I	传送绝对的四字

- 特别注意，对应的movl指令就会将对应的寄存器的高位4字节设置为0
- 为什么会有movabsq
  - 在将对应的立即数作为源操作数时候
  - 常规的movq指令只能以表示32位补码数字的**立即数**作为源操作数，然后把这个值进行对应的符号扩展，再放到对应的目标位置
  - movabsq可以以任意64位**立即数**值作为源操作数，并且只能以寄存器为目的

- 不同的传送方式
  - 零扩展数据传送指令

指令	效果	描述
MOVZ S，R	R←零扩展(S)	以零扩展进行传送
movzbw		将做了零扩展的字节传送到字
movzbl		将做了零扩展的字节传送到双字
movzwl		将做了零扩展的字传送到双字
movzbq		将做了零扩展的字节传送到四字
movzwq		将做了零扩展的字传送到四字

我们发现没有将双字拓展到四字的，为啥？

- 用对应的movl指令就可以实现了
- 符号扩展数据传送指令

指令	效果	描述
MOVS S，R	R←符号扩展(S)	传送符号扩展的字节
movsbw		将做了符号扩展的字节传送到字
movsbl		将做了符号扩展的字节传送到双字
movswl		将做了符号扩展的字传送到双字
movsbq		将做了符号扩展的字节传送到四字
movswq		将做了符号扩展的字传送到四字
movslq		将做了符号扩展的双字传送到四字
cltq	%rax ←符号扩展(%eax)	把%eax 符号扩展到%rax

算术和逻辑操作

- leaq
  - 加载有效地址
  - 从内存读取数据到寄存器，但实际上根本没有引用内存，而只是将对应的有效地址写入到目的操作数
  - leaq的源数据中对应的k一定是1，2，4，8；leaq的目标一定是对应的一个寄存器

- 算术操作

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

- 减法的顺序问题
  - 首先，最后变的是目标数，源数据不变
  - 这个可以考虑对应二元操作数的一些性质，目的数本身就只能是一个寄存器或者一个内存引用
  - 如果目的数是一个立即数，没法存了。但是第一个源数据都可以
  - 且如果对应的目的数是内存，那么最后还要将其写回内存
- 乘法的特殊情况
  - 根据对应操作数的个数，对应汇编器可以自动分辨出使用哪一条指令
  - 当为单操作数时，所做的实际上是全乘法

$$R[\%rdx] : R[\%rax] \leftarrow S \times R[\%rax]$$

其中rdx这个寄存器存储的是高64位，rax存储的是低64位

  - 我们一般用imulq表示对应的有符号数全乘法，mulq表示无符号数全乘法

- 在使用这样的单操作数运算时，先会把另一个数据先放到rax这个寄存器中
- 考虑下面一段代码有

```
x,y--unsigned 64 int
dest in %rdi,x in %rsi,y in %rdx

movq %rsi,%rax
mulq %rdx
movq %rax,(%rdi)
movq %rdx,8(%rdi)

%rdx:%rax
=0x12345ABC 12345ABC 56789CDE 56789CDE

内存中地址为%rdi开始存储的是
DE 9C 78 56 DE 9C 78 56 | BC 5A 34 12 BC 5A 34 12
```

  - 特别注意，这是小端法机器，即低位字节对应的地址小

- 除法
  - 对应的除法将对应寄存器%rdx和%rax中的128位数作为被除数，除数作为指令的操作数，将商存储在rax中，余数存储在rdx中
  - 特别的，当对应的被除数是64位值时，对应就放在rax中，rdx则根据对应的是否有符号运算填充对应的值——此时cqto就有用了
  - 考虑下列代码

```
long q=x/y;
long r=x%y;
```

    我们只看这个运算过程的话，其在对应的汇编语言中就只对应了一句指令idivq

- 移位操作
  - 位操作都是先给出对应的移位量，再给出对应的操作数
  - 移位量可以是一个立即数或者放在单字节寄存器中
  - 移位操作对于w位长的数据值进行操作，移位量是由%cl寄存器低m位决定的，其中2的m次方等于w。

- 我们考虑对应的cl里面全是1，那么对于一个8位的字节来说，就是最多移7位，一个16位的字来说，就是最多移15位
    - 即保证了最多移2的m次方-1位，从而使得对应的位移量小于最大位数