MACHINE PROG: BASICS

Class 9　qzq

编译：

- 编译程序：linux> gcc -Og -o p p1.c p2.c
- 汇编：linux> gcc -Og -S mstore.c（生成.s文件）
- 目标文件：linux> gcc -Og -c mstore.c（生成.o文件）
- 反编译：linux> objdump -d mstore.o（看二进制文件）
- （tips：微软终端用户许可协议中禁止逆向工程）

数据格式

（1 个字=2 字节）

- char : b(bite) 1字节
- short : w(word) 2字节
- int : l(long word), float : s 4字节
- long , *char : q(quarter word), double : l 8字节
- 浮点数使用的指令和寄存器与整数不相同!

操作数指示符

- 立即数：$Imm　　Imm
- 寄存器：r　　R[r]
- 存储器：Imm(r,r*,s)　M[Imm+R[r]+R[r*]·s]
- s = 1,2,4,8，省略时为1
- 汇编器会自动选择最紧凑的方式进行数值编码（能用8位立即数就不用16位）
- 内存引用会根据计算出来的地址（称为有效地址）访问某个内存位置
- 基址和变址寄存器都必须是64位寄存器　例：(%eax) 不合法

一个例子（）

P122，练习题 3.1

- 地址　　值　　操作数值
- 0x100　　0xFF　　%rax
- 0x104　　0xAB　　0x104
- 寄存器　　值　　$0x104
- %rax　　0x100　　(%rax)
- %rdx　　0x3　　4(%rax)

一个例子（）

P122，练习题 3.1

- 地址　　值　　操作数
  值
- 0x100　　0xFF　　%rax　　0x100
- 0x104　　0xAB　　0x104　　0xAB
- 寄存器　　值　　$0x104
  $0x104
- %rax　　0x100　　(%rax)　　0xFF
- %rdx　　0x3　　4(%rax)　　0xAB

指令

注：x in %rdi, y in %rsi, *p in %rdx

- 方法：指令 S, D : D 是目的操作数，S 是源操作数，结果存入 D 中
- mov：赋值　　mov %rdi, %rsi;　// y = x;
- add：加法　　add %rdi, %rsi;　// y = y + x;
- 　　add $0x8, %rdi;　　// x = x + 8;
- 内存引用：使用（）mov (%rdx), %rdi;　// x = *p;
- （%rdx存放的是 p 的地址）

关于指令的注意事项：

*x in %rdi, *y in %rsi, s in %rdx

- D 不能是立即数！
- 一条指令不能完成内存—内存的转换！
- mov (%rdi), %rdx; // s = *x;
- mov %rdx, (%rsi); // *y = s;
- 所有生成32位的mov指令以寄存器为目的时会将高位4字节置零
- 立即数可以补0，内存中的数据可以是任意大小，但寄存器的大小是固定的，因此有寄存器就要看寄存器的字节数

如果两个操作数类型不同呢？

- movs：符号位拓展
- movz：零拓展（不存在 movzlq）
- char—>int: movsbl (%rdi), %rax; movl %eax, (%rsi);
- unsigned char—>long: movzbq(%rdi),%rax;movq%rax,(%rsi);
- int—>char: movl (%rdi), %eax; movb %al, (%rsi);
- movabsq 64位立即数，64位寄存器

关于pushq 和 popq

- pushq 和 popq 经常同时出现
- 栈底是高地址，栈顶是低地址
- pushq %rbp: subq $8, %rsp; movq %rbp, (%rsp);
- popq %rbp: movq (%rsp), %rbx; addq $8, %rsp;

算术和逻辑操作

- 加载有效地址：leaq S, D // D = &S;（不是内存引用！）
- 实际编译时通常用leaq进行优化运算,leaq能执行加法和有限形式的乘法
- 例：leaq 7(%rdx,%rdx,4) = 5x + 7;
- 一元操作的操作数可以是寄存器或内存位置
- 移位操作如果需要用寄存器装移位量，只允许使用%cl寄存器，移位量由%cl寄存器的的低m位决定，其中$2^m = w$ 例：%cl的值为0xFF，指令salb会移7位（因为b是1byte=8位，w=8，m=3，0xFF低三位全为1，对应十进制的7）

特殊算术操作

- 支持128位数的部分操作 %rdx高64位与%rax低64位合体组成128位
- imulq, mulq ： 有/无符号全乘法
- cqto ： 将R[%rax]符号扩展到%rdx中
- 注意：mulq 和 imulq 可以有一个或两个操作数，汇编器可自动分辨
- （两个操作数的情况：两个 64 位数乘积结果保留低 64 位，高 64 位丢弃）
- 存储乘积需要两个movq指令，分别存储低8个字节和高8个字节，在小端法机器中，高位字节储存在大地址

关于除法

idivq S, divq S

- 除法：div无符号除法，idiv有符号除法。
- 使用两个寄存器(%rax，%rdx)和一个操作数来执行，其中%rax表示被除数的低位部分，%rdx表示被除数的高位部分(若没有需要清0)，除数作为指令的操作数给出
- 商存储在%rax中，余数存储在%rdx中
- 有符号除法要用cqto进行符号扩展
- 无符号除法通常寄存器%rdx会事先设置为0（防止贡献额外的高位值）

补充：gdb 调试

- 编译：gcc -g program.c -o program
- 运行 gdb：gdb ./program
- 设置断点：(gdb) break main // 在 main 函数设置断点
- 运行程序：(gdb) run

gdb 调试

- 单步执行：(gdb) step // 进入函数内部
- (gdb) next // 不进入函数，直接进入下一行
- 查看变量的值：(gdb) print value
- 在断点处继续运行：(gdb) continue
- 退出：(gdb) quit

谢谢！