第三次小班课

计算机系统导论 (Class 9)

老师: 汪小林
助教: 陈东武

北京大学 信息科学技术学院

2024 年 09 月 25 日

回课补充

特殊算术指令

x86-64 指令集支持结果为 128 位的乘法 / 被除数为 128 位的除法.

指令	效果	描述
imulq S mulq S	R[%rdx]: R[%rax]←S×R[%rax] R[%rdx]: R[%rax]←S×R[%rax]	有符号全乘法 无符号全乘法
cqto	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[%rax]←R[%rdx]: R[%rax]÷S	有符号除法
divq S	R[%rdx]←R[%rdx]: R[%rax] mod S R[%rax]←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

条件码

- leaq 指令不改变条件码.
- 逻辑操作把 CF 和 OF 设置为 0.
- 移位操作把 CF 设置为最后被移出的位, OF 设置为 0.
- inc 和 dec 指令不会改变 CF.

leaq 指令不改变任何条件码，因为它是用来进行地址计算的。除此之外，图 3-10 中列出的所有指令都会设置条件码。对于逻辑操作，例如 XOR，进位标志和溢出标志会设置成 0。对于移位操作，进位标志将设置为最后一个被移出的位，而溢出标志设置为 0。INC 和 DEC 指令会设置溢出和零标志，但是不会改变进位标志，至于原因，我们就不在这里深入探讨了。

条件传送

- 条件传送指令 `cmovxx`, 适用条件传送的条件:
  - 两个表达式的计算代价都很小.
  - 不会导致错误或副作用.
- https://godbolt.org/ 是一个在线编译器.

使用条件传送也不总是会提高代码的效率。例如，如果 *then-expr* 或者 *else-expr* 的求值需要大量的计算，那么当相对应的条件不满足时，这些工作就白费了。编译器必须考虑浪费的计算和由于分支预测错误所造成的性能处罚之间的相对性能。说实话，编译器并不具有足够的信息来做出可靠的决定；例如，它们不知道分支会多好地遵循可预测的模式。我们对 GCC 的实验表明，只有当两个表达式都很容易计算时，例如表达式分别都只是一条加法指令，它才会使用条件传送。根据我们的经验，即使许多分支预测错误的开销会超过更复杂的计算，GCC 还是会使用条件控制转移。

跳转指令

- 跳转指令中的立即数是 **PC 相对的** ***PC-relative***.
  - 将偏移量 (目标地址 - 跳转指令的下一条指令的地址) 作为编码.
  - 偏移量可以编码为 1 / 2 / 4 字节的**有符号数**.
- 使用相对地址的好处: 编码长度更短 + 代码段不改变就能移动位置.

跳转指令

- 跳转指令中的立即数是 **PC 相对的** *PC-relative*.
  - 将偏移量 (目标地址 - 跳转指令的下一条指令的地址) 作为编码.
  - 偏移量可以编码为 1 / 2 / 4 字节的**有符号数**.
- 使用相对地址的好处: 编码长度更短 + 代码段不改变就能移动位置.
- 练习: 如下跳转指令的目标地址是?
  - `400020: 74 F0        je ______`

控制流图

- **基本块** *Basic Block* 是一组顺序执行的指令.
  - 基本块的入口只有第一条指令, 出口只有最后一条指令.
  - 基本块的中间不能跳转到其他指令, 也不能被其他指令跳转.
  - 基本块的执行具有**原子性** *atomic*: 要么执行完, 要么不执行.
- 划分基本块的方法: 标记出基本块的入口.
  - 包括程序开头, 跳转指令的目标, 跳转指令的下一条指令.

控制流图

- **基本块** *Basic Block* 是一组顺序执行的指令.
  - 基本块的入口只有第一条指令, 出口只有最后一条指令.
  - 基本块的中间不能跳转到其他指令, 也不能被其他指令跳转.
  - 基本块的执行具有**原子性** *atomic*: 要么执行完, 要么不执行.
- 划分基本块的方法: 标记出基本块的入口.
  - 包括程序开头, 跳转指令的目标, 跳转指令的下一条指令.
- 基本块之间的跳转构成了程序的**控制流图** *Control Flow Graph*.
  - 提取程序的控制流信息, 以免被成堆的汇编指令淹没:(
- 控制流图的**支配树** *dominator tree* 上的**返祖边** *back edge* 对应一个循环.
  - 据此得出循环变量等信息.

GDB Tutorial

GDB 简介

- The GNU Debugger (GDB) is a portable debugger that runs on many Unix-like systems and works for **many programming languages**.
- 在编写复杂的程序 (如操作系统、驱动程序或嵌入式系统) 时, 代码出错的概率较大且错误可能难以定位. GDB 可以帮助开发者:
  - **追踪程序的执行**: 逐步执行代码, 查看每一步的状态.
  - **设置断点**: 让程序在某行代码或某个函数停下来, 方便检查运行时的状态.
  - **查看内存和变量**: 实时查看和修改变量的值, 追踪内存中的数据变化.
  - **分析崩溃的程序**: 通过读取 core dump 文件, 分析程序崩溃的原因.
- ~~然而我在打完去年 GeekGame 之后就没用过 GDB 了, 怎么会是呢?~~

GEF 插件

- A modern experience for GDB with advanced debugging capabilities for exploit devs & reverse engineers on Linux.
  - ~~评价是不装插件用裸 GDB 的都是神人子(~~
- 从 https://github.com/hugsy/gef 下载 gef.py, 放到用户目录.
  - 执行 `echo "source ~/gef.py" > ~/.gdbinit`

试看看

- 在用户目录下新建 test.c, 写一个 A/B Problem.
  - 编译命令: `gcc test.c -o test -g -Wall`
  - 如果只想在汇编代码上调试, 就删除 `-g` 选项.
  - 添加优化选项 `-O2` 可能会使汇编代码更简洁, 但不方便在源代码上调试.
- 在 .gdbinit 添加 `set disassembly-flavor att` 将汇编代码格式改为 AT&T.
- 执行 `gdb test` 启动 GDB, 指定 test 作为要调试的目标文件.
- 各种命令参见 https://pare1lel.github.io/ics-fa24/assets/gdbnotes-x86-64.pdf
- 好的你已经会用 GDB 了, 该去 “炸弹实验室” 了.

Exercise

- 以下关于数据传送指令的说法中, 正确的是?
  - A. `movq` 只能以 32 位立即数作为源操作数, 然后将这个值零扩展到 64 位;
    `movabsq` 能够以 64 位立即数作为源操作数, 但只能以寄存器为目的.
  - B. 在 `Imm(rb,ri,s)` 这一寻址模式中, s 必须是 1 / 2 / 4 / 8;
    基址和变址寄存器必须是 64 位寄存器.
  - C. `cqto` 指令不需要额外操作数, 它的作用是将 `%eax` 符号扩展到 `%rax`.
  - D. 执行指令 `movl -1, %eax` 后, `%rax` 的值为 `0x00000000ffffffff`.

- 以下关于数据传送指令的说法中, 正确的是?
  - A. `movq` 只能以 32 位立即数作为源操作数, 然后将这个值零扩展到 64 位;
    `movabsq` 能够以 64 位立即数作为源操作数, 但只能以寄存器为目的.
  - B. 在 `Imm(rb,ri,s)` 这一寻址模式中, s 必须是 1 / 2 / 4 / 8;
    基址和变址寄存器必须是 64 位寄存器.
  - C. `cqto` 指令不需要额外操作数, 它的作用是将 `%eax` 符号扩展到 `%rax`.
  - D. 执行指令 `movl -1, %eax` 后, `%rax` 的值为 `0x00000000ffffffff`.

- 在下面的问号上填空:
  - A. `mov? (%rdx), %rax`
  - B. `mov? %dl, (%rsp,%rdx,4)`
  - C. `mov? %eax, (%rsp)`
  - D. `mov? (%rax), %dx`

- 在下面的问号上填空:
  - A. `movq (%rdx), %rax` B. `movb %dl, (%rsp,%rdx,4)`
  - C. `movl %eax, (%rsp)` D. `movw (%rax), %dx`

- 在下面的问号上填空:
  - A. `movq (%rdx), %rax`　　B. `movb %dl, (%rsp,%rdx,4)`
  - C. `movl %eax, (%rsp)`　　D. `movw (%rax), %dx`
- 以下哪些指令在 x86-64 指令集下是不合法的?
  - A. `movb $0xF, (%ebx)`　　B. `movl %rax, (%rsp)`
  - C. `movw (%rax), 4(%rsp)`　　D. `movb %al, %sl`

- 在下面的问号上填空:
  - A. `movq (%rdx), %rax`　　B. `movb %dl, (%rsp,%rdx,4)`
  - C. `movl %eax, (%rsp)`　　D. `movw (%rax), %dx`
- 以下哪些指令在 x86-64 指令集下是不合法的?
  - A. `movb $0xF, (%ebx)`　　B. `movl %rax, (%rsp)`
  - C. `movw (%rax), 4(%rsp)`　　D. `movb %al, %sl`

- 课本练习题 3.21.

#thanks