语义实验

2024.4.23

实验目标

- 在语法实验的基础上，修改语义动作，输出SysY代码对应的x86-64汇编代码。

实验指导

样例

```c
#include <stdio.h>
const int c = 0x12;
int d[5], e;

int neg(int k)
{
    return -k;
}

int main()
{
    int a = 1, b = 2;
    d[3] = 4;
    d[2] = (a + c) * (d[3] - b) - a * b;
    d[4] = neg(d[2] + d[3]);
    scanf("%d", &e);
    if(d[4] == e)
        printf("yes!\n");
    else
        printf("Your answer %d is wrong!\n", e);
    return 0;
}
```

```
.intel_syntax noprefix

.section .rdata
.LC0:
        .string "%d"
.LC1:
        .string "yes!"
.LC2:
        .string "Your answer %d is wrong!\n"

c:      .long   0x12

.data
d:      .zero   20
e:      .zero   4

.text
.globl          main
.globl          neg

neg:
        push    rbp
        mov     rbp, rsp
        mov     dword ptr [rbp-4], ecx
        mov     eax, dword ptr [rbp-4]
        neg     eax
        leave
        ret
```

```
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        mov     dword ptr [rbp-4], 1
        mov     dword ptr [rbp-8], 2
        mov     dword ptr d[rip+12], 4
        mov     eax, dword ptr [rbp-4]
        add     eax, dword ptr c[rip]
        mov     edx, eax
        mov     eax, dword ptr d[rip+12]
        sub     eax, dword ptr [rbp-8]
        imul    edx, eax
        mov     eax, dword ptr [rbp-4]
        imul    eax, dword ptr [rbp-8]
        sub     edx, eax
        mov     dword ptr d[rip+8], edx
        mov     edx, dword ptr d[rip+8]
        mov     eax, dword ptr d[rip+12]
        add     eax, edx
        mov     ecx, eax
        call    neg
        mov     dword ptr d[rip+16], eax
        lea     rdx, e[rip]
        lea     rcx, .LC0[rip]
        call    scanf
        mov     edx, dword ptr d[rip+16]
        mov     eax, dword ptr e[rip]
        cmp     edx, eax
        jne     .L4
        lea     rcx, .LC1[rip]
        call    printf
        jmp     .L5
.L4:
        mov     eax, dword ptr e[rip]
        mov     edx, eax
        lea     rcx, .LC2[rip]
        call    printf
.L5:
        mov     eax, 0
        leave
        ret
```

GNU汇编语法

- 基础知识（自行回顾ICS1所学的逆向工程基础）
  - 汇编指令格式
  - 数据传送与计算
  - 条件处理
  - 过程
- 进阶知识
  - 符号与伪指令
  - 段分配
  - 元素处理
  - 环境处理

符号

- 符号是程序的一个核心概念：程序员使用符号来命名事物，链接器使用符号进行链接，调试器使用符号进行调试。
- label：标签(label) 被写成一个符号，其后紧跟一个冒号，这种符号表示活动位置计数器的当前值。例如样例中main:表示其第一条指令“push rbp”的指令地址。
- global：全局符号(global symbol)可以使用 .global 来声明。全局符号可以被其他模块引用。 例如样例中neg、main函数是外部函数。
- 本地符号：主要是以某些本地标签前缀开头的任何符号，在本地汇编代码中定义和使用。默认情况下，本地标签为 .Lx，且本地符号不会出现在符号表中，因此它们在调试时不可见。例如样例中的.L4、.L5。

伪指令

- 数据定义伪指令：
  - .byte ： 把8 位数当成数据插入到汇编中。
  - .hword： 把16位数当成数据插入到汇编中。
  - .long和 .int：把32位数当成数据插入到汇编中。
  - .quad： 把64位数当成数据插入到汇编中。
  - .float: 把浮点数当成数据插入到汇编中。
  - .ascii “string”：把 string 当作数据插入到汇编中，ascii 伪操作定义的字符串需要自行添加结尾字符\0。
  - .asciz “string” 和.string： 类似 ascii, 在string 后面插入 一个结尾字符 \0。
  - .rept 和 .endr: 重复执行伪操作。

段分配

- .section 伪指令表示接下来的汇编会链接到某个段，例如代码段、数据段等，其格式如下，name 表示段的名称：flags 表示段的属性：
  - .section name, "flags"
  - flags可为如下属性（若不存在flags，则表示该段为只读）：

属性	说明
a	段具有可分配属性
d	段具有 GNU_MBIND属性的段
e	段被排除在可执行和共享库之外
w	段具有可写属性
x	段具有可写执行属性
M	段具有可合并属性
S	段包含零终止字符串
G	段是段组(section group)的成员
T	段用于线程本地存储(thread-local-storage)

段分配

- 常见段的作用：
  - .text（代码段，权限可读可执行）
    - 通常用于存储运行指令，无需用.section声明
  - .data（数据段，权限可读可写）
    - 通常用于存储全局变量，无需用.section声明
  - .rdata（只读数据段，权限可读）
    - 通常用于存储常量，例如字符串、跳转表等

元素处理

- 符号表定义：
  - 符号表是编译器中的一个重要组成部分，用于存储程序中的各种符号及其相关信息，以便在编译过程中进行查找和管理。

- 符号表作用：
  - 存储符号的名称、类型、作用域、生命周期等信息；
  - 提供查找符号的接口，以便在编译过程中进行查找；
  - 管理符号的生命周期，确保符号在其作用域内有效。

- 符号表实现：
  - 基于哈希表的符号表实现主要包括以下步骤：
    - 创建一个哈希表，用于存储符号。
    - 当遇到一个新的符号时，将其名称和相关信息插入到哈希表中。
    - 当需要查找一个符号时，使用其名称进行哈希计算，然后在哈希表中查找。
    - 当需要删除一个符号时，将其从哈希表中删除。
    - 哈希表的实现可以使用数组、链表、树等数据结构，常用的实现方式有：
      - 数组实现：使用数组存储哈希表，通过计算哈希值来查找符号。
      - 链表实现：使用链表存储哈希表，当数组中的某个位置已经存储了多个符号时，使用链表连接这些符号。
      - 树实现：使用树存储哈希表，当数组中的某个位置已经存储了多个符号时，使用树连接这些符号。

元素处理

- 常量：
  - 对于字符串，定义在.rdata段（需使用.section声明），如样例：
    - .LC1: .string "yes!"
  - 对于整型数据，可以定义在.rdata段，也可以在编译时直接确定该常量的值，在使用该常量的地方，把该常量直接换成对应的数。

元素处理

- 局部变量：
  - 使用栈或者寄存器存储，推荐使用寄存器存储（耗费时间更少）。
  - 注意变量的类型大小，使用合适的伪指令（byte ptr、dword ptr）获取内存数据，特别注意对64位寄存器低32位赋值会导致其高32位清零。

- 全局变量：
  - 使用方式：使用.sym[rip]表示该变量内容，其中sym表示该全局变量名称。
  - 例：.LC0[rip]（字符串.LC0内容）、e[rip]（全局变量e内容）
  - 特别注意，若需要取变量地址，需使用lea指令，如样例中的：

```
lea    rdx, e[rip]          // &e
lea    rcx, .LC0[rip]       // "%d"
call   scanf                // scanf("%d", &e);
```

元素处理

- 数组：
  - 对于数组定义，此时括号内部都是常数，直接计算即可。在匹配的时候可以记录数组每一维的维数，同时统计数组所需要占据的空间大小。
  - 对于数组的使用，需要递归从最后一维开始计算该变量所在的地址。需要注意的是，这时候不一定每一维都是常数，不能直接通过计算得到，需要一步一步使用汇编实现地址的计算。
  - 数组作为函数参数传递，本质是传递数组地址，可以直接在符号表中进行记录，而无需再汇编指令中进行传递。

元素处理

- 函数调用
  - 对于库函数printf、scanf等，直接call printf/scanf即可。其调用方式为__cdecl，需要调用函数清理堆栈，调用时rsp需对齐16字节，传递参数时需遵守系统环境的要求。调用库函数时，需自行保存caller-saved寄存器，例如调用printf前需使用了rbx则需自行保存，否则调用后该寄存器的值将不正确。
  - 对于自定义函数，可以自行选择调用方式，推荐使用__stdcall，即被调用函数清理堆栈，这样保证了堆栈始终是平衡的。自定义函数的传递参数可自行设计。
  - 进入一个函数时，建议遵循x86-64调用约定，保存所有callee-saved寄存器（rbp等），在退出函数时还原这些寄存器并恢复栈帧。建议在语义分析时计算得到函数内需要多少空间，并在进入函数时直接分配相应空间。

元素处理

- 控制流语句：
  - 对于if语句，在条件判断的时候处理出truelist和falselist，然后在if的分支中打标记，使用回填的方法和处理truelist和falselist中指令的跳转位置。如样例中的.L4和.L5。
  - 对于while语句，由于其判断语句可能会执行多次，所以在进入判断语句之间就要对栈中的内容进行维护和对齐，退出时要恢复到原来的样子。
  - 对于break和continue语句，需要要求其在while内才能使用。对于这两条语句，需要先对栈中内容进行恢复，才能进行跳转。否则最后程序结束时栈顶指针无法恢复到原来的状态。同时，需要记录break和continue针对的是哪个循环语句，需要弹出栈顶多少的内容。
  - 由于break和continue也需要跳转到循环的尾部，而跳转的时候循环尾部还没进行标记，所以需要在标记之后进行回填。
  - 注意条件分支前后对符号表的修改和对栈平衡的处理。

环境处理

- Windows：
  - 若一个函数调用了库函数，则进入该函数时，需至少分配32字节的栈空间。
  - 调用库函数时，前4个参数是rcx、rdx、r8、r9，后续参数用栈传递。
  - 在调用库函数时，rsp必须对齐16字节边界。

- Linux
  - 调用库函数时，前6个参数是rdi、rsi、rcx、rdx、r8、r9，后续参数用栈传递。
  - 在调用库函数时，rsp必须对齐16字节边界。

实验测评

- 本实验使用的源文件会在SysY文法上略微进行修改，以.c后缀保存，方便同学们直接用gcc生成对应汇编代码进行对比。
- 本实验后续会下发若干源文件和对应的运行结果作为平时测试样例，最终验收时会现场下发5个（暂定）同类别的考试样例，该实验最终分数将主要取决于你的考试样例得分。
- 鼓励同学间相互交流，严禁抄袭代码，若查重发现将会使抄袭者、被抄袭者该实验得分为0，同时影响抄袭者其它实验分数。
- 请善于使用IDA、x64dbg、pwndbg(gdb)、Compiler Explorer等工具进行分析和调试。

参考资料

- https://gcc.godbolt.org/
- https://blog.csdn.net/daocaokafei/article/details/115439936
- https://blog.51cto.com/u_7090376/1264642