ICS Seminar Week4 Prep

余文凯 康子熙 赵廷昊 许珈铭

2023.10.7

Q1

9. pushq %rbp 的行为等价于以下（）中的两条指令。

```
A. subq $8, %rsp    movq  %rbp,  (%rdx)
B. subq $8, %rsp    movq  %rbp,  (%rsp)
C. subq $8, %rsp    movq  %rax,  (%rsp)
D. subq $8, %rax    movq  %rbp,  (%rdx)
```

pushq指令的作用是向栈中压入数据，由于栈是从大地址向小地址拓展的，所以应当先把栈顶的指示地址（也就是%rsp中存储的值）减8，然后再向栈顶的指示地址对应的内存中存入数据。

B

Q2

6. 有如下代码段：

```
int func(int x, int y);
int (*p) (int a, int b);
p = func;
p(0,0);
```

对应的下列 x86-64 过程调用正确的是：

A. call *%rax　　B. call (%rax)

C. call *(%rax)　　D. call func

%rax和(%rax)都可以起到寻址的作用，但是一般(%rax)会使用在内存访问时，而%rax会用在间接跳转(jmp)和调用(call)时。所以A选项和D选项都是可选的选项。
注意，在x86-64机器上，函数的地址会加载到%rax中，这和参数存储在%rdi，%rsi等寄存器上并不一样。

A/D

Q3

5. 已知函数 func 的参数超过 6 个。当 x86-64 机器执行完指令 call func 之后，%rsp 的值为 S。那么 func 的第 k(k > 6)个参数的存储地址是？

A. S + 8 * (k - 6)

B. S + 8 * (k - 7)

C. S - 8 * (k - 6)

D. S - 8 * (k - 7)

在call func后，压栈的不只有func所传的参数，同时还有函数的返回地址（也就是下一条指令的地址）。同时，由于参数传递是从右向左传递的，所以在栈上，更靠前的参数应该存在地址更低的地方，且栈顶存储的是函数的返回地址。如此一来，第七个参数应该储存在S+8的地方，通过排除法，可以找到正确的答案。

A

Q4

4、下列关于 x86-64 过程调用的叙述中，哪一个是不正确的？

A．每次递归调用都会生成一个新的栈帧，空间开销大

B．当传递给被调用函数的参数少于 6 个时，可以通过通用寄存器传递

C．被调用函数要为局部变量分配空间，返回时无需释放这些空间

D．过程调用返回时，向程序计数器中推送的地址是调用函数中调用指令的下一条指令的地址

C错误，其原因是被调用函数为局部变量分配空间时，由于在调用函数中，那几个被调用者保存的寄存器可能被使用了，所以需要把这些寄存器中的值存储到栈上，再给被调用函数使用，且被调用函数使用完需要重新加载这些在栈上的参数。

C

Q5

7. 考虑以下 C 语言变量声明：

```
int *(*f[3])();
```

那么在一台 x86-64 机器上，sizeof(f)和 sizeof(*f)的值是多少？

A. 8 24

B. 24 8

C. 8 8

D. 8 不确定

一道指针的阅读题。通过阅读指针，可以看出来外层是一个函数指针，内层是一个指针数组，所以整个变量应该代表了一个函数指针的数组，且这些函数都不接受任何参数并返回一个int。所以sizeof(f)应当是指针数组的大小，即8*3=24。而sizeof(*f)是取数组的第一个元素，指的是一个函数指针的内存开销，也就是8个字节。

B

Q6

7. 有A的定义：int A[3][2] = {{1,2}, {3,3}, {2,1}};
那么A[2]的值为：

A. &A+16　　B. A+16　　C. *A+4　　D. *A+2

选项A，在C++中，引用的本质是数组，所以可以将&A理解成int (*)A[3][2],即是一个指向高维数组的指针。此时在执行+16操作时，默认进行的是指针的常数偏移操作，即会偏移总共16*8个字节，显然不是要找的地址。
选项B，A可以指代数组的第一个元素，使用A+16寻找的是数组的第17个元素，显然不是要找的地址。
选项C，首先判断优先级，*的优先级高于加法，*A可以理解成对外层数组的一个解地址，应当得到A[0]。此时这个新的数组的内存占用应当是4*2=8，故*A+4应当是X+4*8=X+32，正是我们要找的目标地址。
选项D，与选项C同理，应当找到的是X+16而非X+32，错误。

C

Q7

3、假定静态 int 型二维数组 a 和指针数组 pa 的声明如下：

```
static int a[4][4]={ {3, 8, -2, 6}, {2, 1, -5, 3 }, {1, 18,
4, 10},{4, -2, 0, 8}};
static int *pa[4]={a[0], a[1], a[2], a[3]};
```

若 a 的首地址为 0x601080，则&pa[0]和 pa[1]分别是：

A. 0x6010c0、0x601090

B. 0x6010e0、0x601090

C. 0x6010c0、0x6010a0

D. 0x6010e0、0x6010a0

注意到pa是在a之后定义的变量，a的总长度为4*4*4=64字节，故pa的初始地址应当是0x6010c0，即&pa[0]。
同时，观察到pa[1]中存储的是a[1]，且pa是一个指针类型的变量，所以pa[1]应当是a[1]对应的地址，也就是0x601090。

A