ICS-07 Data

By 胡仕豪

数组

一维数组

嵌套数组

多层数组

异质

数据对齐

结构体与联合

数组 一维数组

1、在内存中分配连续空间进行**存储**，大小为单元数据类型大小*数组大小

2、调用过程**数据传送**时，传送或返回数组第一个元素的地址

3、**寻址**过程满足公式 $x_i = x_A + L \cdot i$

数组 嵌套数组

1、区别仅在于将一维数组的单元换为一维数组

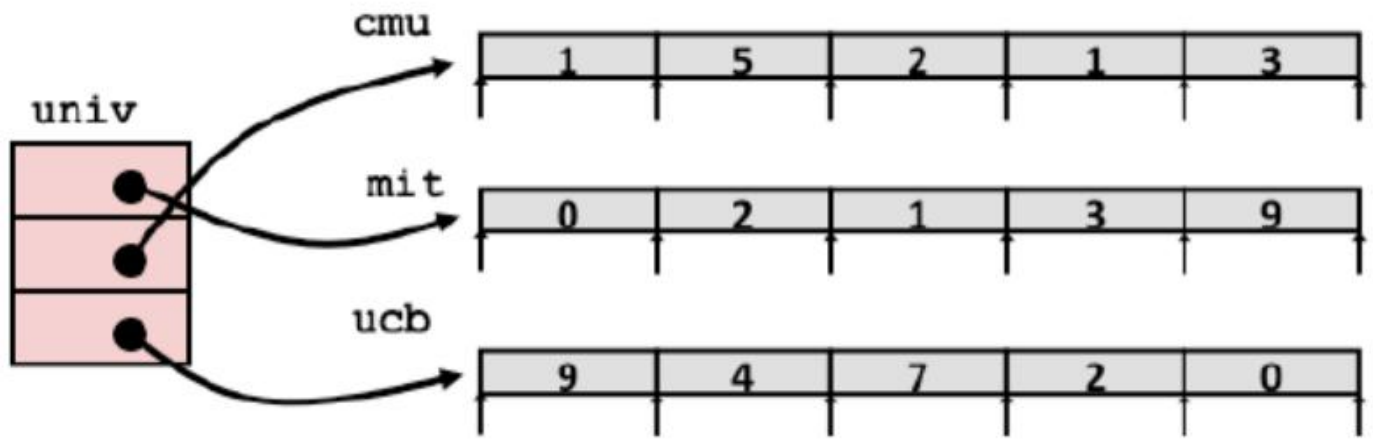
2、寻址过程满足公式&D$[i][j] = x_D + L(Ci + j)$

```
leaq (%rsi,%rsi,x),%rax    // 计算Ci，C值不同计算方式可能也不尽相同。
leaq (%rdi,%rax,L),%rax    // 计算xD+LCi
movq (%rax,&rdx,L),%rax    // 得到内存中xD+LCi+Lj的内容。
```

3、嵌套数组的定长与否决定了计算Ci的时间消耗

数组 多层数组

1、用一个一维数组**存储**行向量的地址



2、寻址过程满足公式$\&D[i][j] = \mathrm{Mem}[\mathrm{Mem}[x_D + 8i] + Lj]$

3、多层数组优化了修改行向量长度的复杂性

异质 数据对齐

1、为优化读取时间，系统建议K字节的基本对象，其地址必须是K的倍数

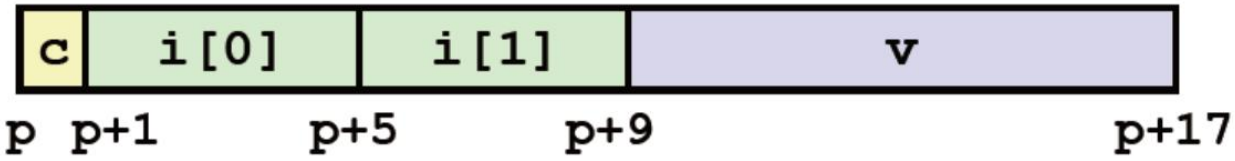
2、对于数组、结构这种复合类型，K取其中基本对象中占字节数最大值

3、注意强制对齐的情况：有些实现SSE的指令需要数据进行16字节对齐

异质 结构体

1、结构体的存储顺序按照声明顺序存放

- **Unaligned Data**

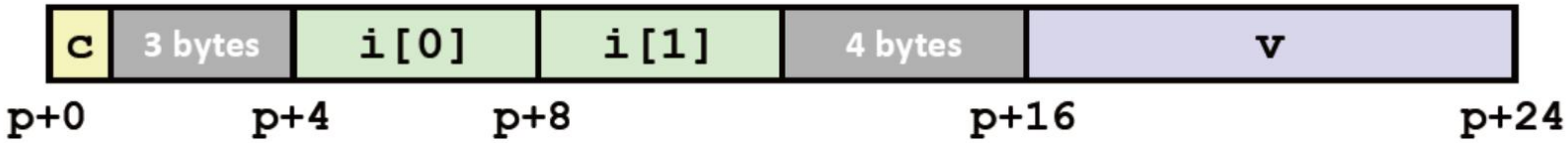



```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

2、结构体尾部需填充间隙使其长度为K的倍数

3、结构体基本单元满足数据对齐要求偏移

- **Aligned Data**



异质 联合

1、联合存储长度为其基本单元中最大者向上取至K的倍数

2、各个基本单元不存在偏移，而是使用不同方式解释相同内存字段

```
double uu2double(unsigned w0, unsigned w1){    // 按位转换成双精度浮点数
    union {
        double d;
        unsigned u[2];
    } t;

    t.u[0] = w0;
    t.u[1] = w1;
    return t.d;
}
```

数组 一维数组

连续定义**相同类型**的数组地址连续

```
#define ZLEN 3
typedef int z[ZLEN];

int check(){
    z happy = {1, 2, 3};
    z yummy = {4, 5, 6};
    z tasty = {7, 8, 9};
```

```
gef➤  x/32x $rsp
0x7fffffffdde0: 0x00000000      0x00000001      0x00000002      0x00000003
0x7fffffffddf0: 0x00000004      0x00000005      0x00000006      0x00000007
0x7fffffffde00: 0x00000008      0x00000009      0xc978b700      0xf8b833ae
0x7fffffffde10: 0xffffde30      0x00007fff      0x555551f4      0x00005555
0x7fffffffde20: 0xffffdf10      0x00007fff      0xffffdf58      0x00007fff
0x7fffffffde30: 0xffffded0      0x00007fff      0xf7c2a1ca      0x00007fff
0x7fffffffde40: 0xffffde80      0x00007fff      0xffffdf58      0x00007fff
0x7fffffffde50: 0x55554040      0x00000001      0x555551de      0x00005555
```

happy[3]
yummy[3]
tasty[3]

数组 一维数组

连续定义**不同类型**的数组地址相对顺序可能被编译器优化

```
typedef int z[ZLEN];
typedef char y[ZLEN];
typedef float x[ZLEN];

int check(){
    z happy = {1, 2, 3};
    y yummy = {'a', 'b', 'c'};
    x tasty = {1.0, 2.0, 4.0};
    return 0;
```

```
gef➤  x/32x $rsp
0x7fffffffdde0: 0x00000000      0x00000000      0x00000000      0x00000001
0x7fffffffddf0: 0x00000002      0x00000003      0x3f800000      0x40000000
0x7fffffffde00: 0x40800000      0x63626100      0xec09b300      0x7140b438
0x7fffffffde10: 0xffffde30      0x00007fff      0x555551fb      0x00005555
0x7fffffffde20: 0xffffdf10      0x00007fff      0xffffdf58      0x00007fff
0x7fffffffde30: 0xffffded0      0x00007fff      0xf7c2a1ca      0x00007fff
0x7fffffffde40: 0xffffde80      0x00007fff      0xffffdf58      0x00007fff
0x7fffffffde50: 0x55554040      0x00000001      0x555551e5      0x00005555
```

happy[3]
tasty[3]
yummy[3]

数组 一维数组

连续定义**相同类型**的数组地址连续

```c
#include <stdio.h>
#define ZLEN 5
typedef int z[ZLEN];

void check(){
    z happy = {1, 2, 3, 4, 5};
    z yummy = {6, 7, 8, 9, 10};
    z tasty = {11, 12, 13, 14, 15};
}
```

```
gef➤  x/32x $rsp
0x7fffffffddf0: 0x00000001      0x00000002      0x00000003      0x00000004
0x7fffffffde00: 0x00000005      0x00000001      0x00000000      0x00000000
0x7fffffffde10: 0x00000006      0x00000007      0x00000008      0x00000009
0x7fffffffde20: 0x0000000a      0x00000000      0x00000000      0x00000000
0x7fffffffde30: 0x0000000b      0x0000000c      0x0000000d      0x0000000e
0x7fffffffde40: 0x0000000f      0x00000000      0xc91f1d00      0x74aa958b
0x7fffffffde50: 0xffffde60      0x00007fff      0x555551f6      0x00005555
0x7fffffffde60: 0xffffdf00      0x00007fff      0xf7c2a1ca      0x00007fff
```

happy[5]

yummy[5]

tasty[5]

异质 数据对齐

注意：考虑到强制对齐的情况，编译器可能会让**不在复合结构内**的**超过16字节的复合结构**按16字节对齐。

```
struct S1{
    int a;
    long b;
    float c;
    char d;
};
```

```
int check(){
    struct S1 happy = {1, 1, 1.0, 'a'};
    struct S1 yummy = {2, 2, 2.0, 'b'};
    struct S1 tasty = {4, 4, 4.0, 'c'};
```

```
gef➤  x/32x 0x7fffffffddb0
0x7fffffffddb0: 0x00000001      0x00000000      0x00000001      0x00000000
0x7fffffffddc0: 0x3f800000      0x00000061      0xffffddf8      0x00007fff
0x7fffffffddd0: 0x00000002      0x00000001      0x00000002      0x00000000
0x7fffffffdde0: 0x40000000      0x00000062      0x00000000      0x00000000
0x7fffffffddf0: 0x00000004      0x00000000      0x00000004      0x00000000
0x7fffffffde00: 0x40800000      0x00000063      0x00000000      0x00000000
0x7fffffffde10: 0xffffde30      0x00007fff      0x555551ce      0x00005555
0x7fffffffde20: 0xffffdf10      0x00007fff      0xffffdf58      0x00007fff
```

练习

如果编译器将**不在复合结构内**的**超过16字节的复合结构**按16字节对齐，且下列代码输出结果第一行为0x7fffffffdd40,则输出结果第三行为？

A、0x7fffffffddcd

B、0x7fffffffddd4

C、0x7fffffffddf4

D、0x7fffffffde0d

```c
#include <stdio.h>
typedef struct S1{
    int a[5];
    char b;
} z;

void check(){
    z happy[2];
    z yummy[3];
    z tasty[4];
    printf("%p\n",happy);
    printf("%p\n",yummy);
    printf("%p\n",&tasty[0].b);
}
int main () {
    check();
    return 0;
}
```

谢谢