作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师
https://github.com/drink-cat/Book_Program-Principles

从汇编角度总结 C 语言的高效

C 语言非常高效。这里从汇编角度分析其中的秘密。
C 语言贴近汇编语言。数据类型、变量定义、指令运算等方面，相似度高。
C 语言代码编译为汇编代码，转化率高，冗余代码少。
C 语言的指针操作近似于汇编语言的地址操作。指针读写内存，指令精简，简单直接。

比较数据类型(举例)
```
int32_t 对应 .long
int64_t 对应 .quad
float 对应 .float
double 对应 .double
```

比较变量定义(举例)
`int32_t aa = 200` 对应 `aa : .long 200` 。
`double bb = 33.333` 对应 `bb : .double 33.333` 。
`int32_t arr[3] = {100,200,300}` 对应 `arr : .long 100,200,300` 。这里也可以把.long 写成多行。
`int32_t *ptr = &aa` 对应 `ptr : .quad aa` 。

比较类型转换(举例)
16 位整数转 32 位整数， `int32 = (int32_t) int16` 对应 `movswl %bx, %ecx` 。
64 位整数转 64 位浮点数， `float64 = (double) int64` 对应 `cvtsi2sd %rbx, %xmm2` 。

比较指令运算(举例)
32 位整数加法运算， `cc = aa + 5` 对应 `movl aa(%rip), %ecx` 、 `addl $5, %ecx` 。
64 位整数自增运算， `bb++` 对应 `incq %rbx` 。
64 位浮点数乘法运算， `f2 = f1 * f2` 对应 `mulsd %xmm1, %xmm2` 。
16 位整数比较运算， `if( cc >= dd )` 对应 `cmpw %edx, %ecx` 、 `jge tmp_ge` 。

用 C 和汇编分析高效的细节

编写代码：fast.c
```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

struct cat
{
    int32_t cat_age;
    int32_t cat_speed;
```

```
};

// 整数
int32_t int32_aa = 0x11223344;
int32_t int32_bb = 0x55667788;
// 数组
int32_t int32_arr[2] = {0xCCCCCCCC, 0xDDDDDDDD};
// 结构体
struct cat cat_tom = {.cat_age = 0xA1A2A3A4, .cat_speed = 0xB1B2B3B4};

// ---------------------------------
void func_bird_fly(int height)
{
    printf(" func Bird is flying at height %d \n", height);
}

void func_cat_run(int speed)
{
    printf(" func Cat is running at speed %d \n", speed);
}

void (*func_ptr)(int param);

void print_int32(char *desc, int32_t *ptr)
{
    // 指针转为地址
    uint64_t addr = (uint64_t)ptr;
    // 指针读数据
    int32_t val = *ptr;
    printf(" %15s   addr = %llu   value = %#X  \n", desc, addr, val);
}

int main()
{
    // 打印地址、值
    print_int32("int32_aa", &int32_aa);
    print_int32("int32_bb", &int32_bb);
    print_int32("int32_arr[0]", &(int32_arr[0]));
    print_int32("int32_arr[1]", &(int32_arr[1]));
    print_int32("cat_age", &(cat_tom.cat_age));
    print_int32("cat_speed", &(cat_tom.cat_speed));

    printf("\n === after modify int32 ===\n\n");

    // 指针
    int32_t *num_ptr;
    // 修改整数变量
    num_ptr = &int32_aa;
    *num_ptr = 0xAABBCCDD;
```

```
    // 修改结构体变量的属性
    num_ptr = &(cat_tom.cat_speed);
    *num_ptr = 0xC1C2C3C4;
    // 读指针
    int32_bb = *num_ptr;

    print_int32("int32_aa", &int32_aa);
    print_int32("cat_speed", &(cat_tom.cat_speed));

    printf("\n === see func pointer ===\n\n");

    // 调用函数
    func_ptr = func_bird_fly;
    func_ptr(200);
    func_ptr = func_cat_run;
    func_ptr(300);

    return 0;
}
```

编译代码：

```
gcc fast.c  -S -o fast.s
gcc fast.c  -o fast
```

运行代码：

```
[root@local fast_c]# ./fast
        int32_aa   addr = 6295612   value = 0X11223344
        int32_bb   addr = 6295616   value = 0X55667788
    int32_arr[0]   addr = 6295620   value = 0XCCCCCCCC
    int32_arr[1]   addr = 6295624   value = 0XDDDDDDDD
         cat_age   addr = 6295628   value = 0XA1A2A3A4
       cat_speed   addr = 6295632   value = 0XB1B2B3B4

 === after modify int32 ===

        int32_aa   addr = 6295612   value = 0XAABBCCDD
       cat_speed   addr = 6295632   value = 0XC1C2C3C4

 === see func pointer ===

 func Bird is flying at height 200
 func Cat is running at speed 300
```

分析结果：
变量 int32_aa、int32_bb、int32_arr、cat_tom，以 4 个字节为单位，输出地址和值。
地址依次为 6295612、6295616、6295620、6295624、6295628、6295632。
地址连续，依次相差 4 个字节。内存布局紧凑，内存利用率高。

取地址，使用 mov 指令，指令数少，简单直接。

C 语言代码 `&int32_aa` 对应汇编代码 `movl  $int32_aa, %esi` 。
C 语言代码 `&int32_bb` 对应汇编代码 `movl  $int32_bb, %esi` 。
C 语言代码 `&(int32_arr[0])` 对应汇编代码 `movl  $int32_arr, %esi` 。
C 语言代码 `&(int32_arr[1])` 对应汇编代码 `movl  $int32_arr+4, %esi` 。
C 语言代码 `&(cat_tom.cat_age)` 对应汇编代码 `movl  $cat_tom, %esi` 。
C 语言代码 `&(cat_tom.cat_speed)` 对应汇编代码 `movl  $cat_tom+4, %esi` 。

指针读写内存，使用 mov 指令，指令数少，简单直接。
C 语言代码 `num_ptr = &(cat_tom.cat_speed)` 对应汇编代码 `movq  $cat_tom+4,  -8(%rbp)` 、 `movq  -8(%rbp), %rax` 。
C 语言代码 `*num_ptr = 0xC1C2C3C4` 对应汇编代码 `movl  $-1044200508, (%rax)` 。
C 语言代码 `int32_bb = *num_ptr` 对应汇编代码 `movq  -8(%rbp), %rax` 、 `movl  (%rax), %eax` 、 `movl  %eax, int32_bb(%rip)` 。

函数调用，使用 mov、call 指令，指令数少，简单直接。
C 语言代码 `func_ptr = func_bird_fly` 对应汇编代码 `movq  $func_bird_fly, func_ptr(%rip)` 。
C 语言代码 `func_ptr(200)` 对应汇编代码 `movq  func_ptr(%rip), %rax` 、 `movl  $200, %edi` 、 `call  *%rax` 。

C 语言函数，对应汇编函数。如果开启编译优化，汇编代码会更加简洁。
C 语言函数为

```
void func_cat_run(int speed)
{
    printf(" func Cat is running at speed %d \n", speed);
}
```

对应的汇编函数为

```
func_cat_run:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $16, %rsp
    movl    %edi, -4(%rbp)
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    movl    $.LC1, %edi
    movl    $0, %eax
    call    printf
    leave
    ret
```