常用语法

@M了个J

https://weibo.com/exceptions

https://github.com/CoderMJLee

实力IT教育 www.520it.com

小码哥教育 SEEMYGO

基础知识

■ 学习条件

□ 看过课程的前2节试学视频（https://ke.qq.com/course/336509）

□ 熟悉C语言（M了个J博客）

□ 熟悉任何一门面向对象语言（比如Java、Objective-C、PHP、JavaScript等）

■ 建议

□ 用1.5~2倍速度复习

□ 尊重他人

■ 语法须知

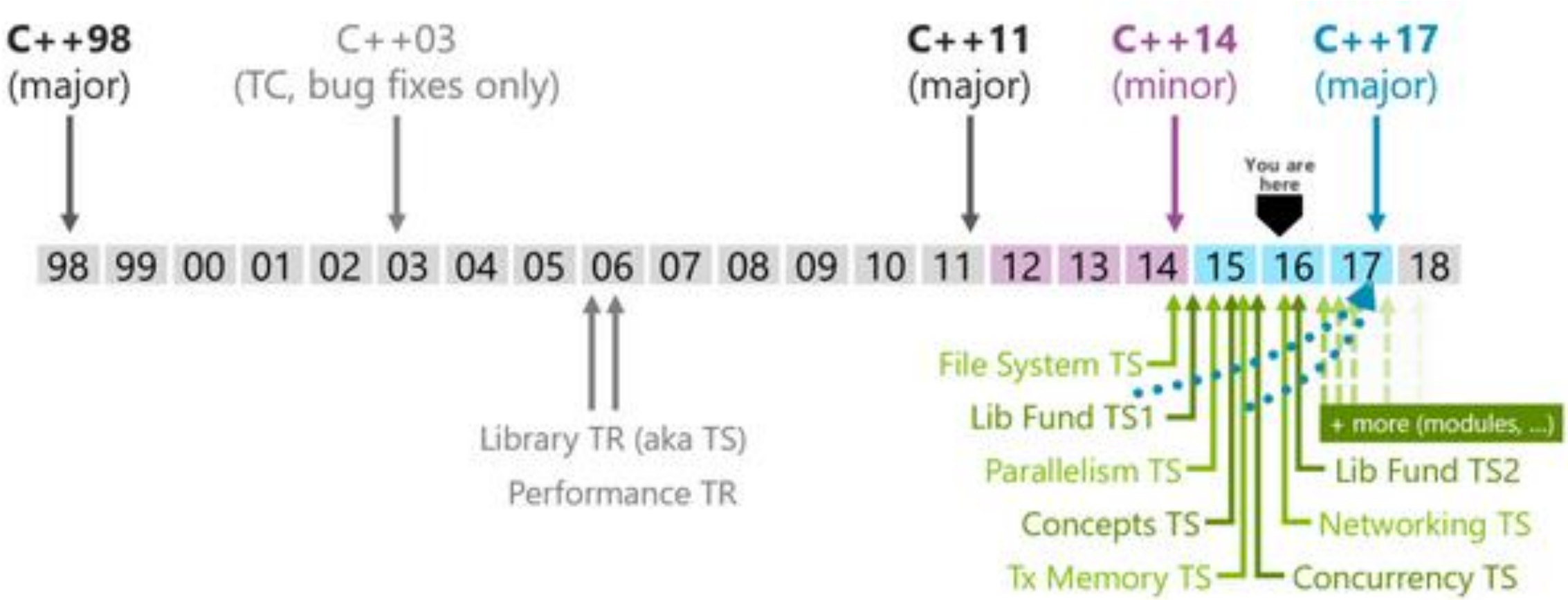
□ C++的源文件扩展名是：cpp（c plus plus的简称）

□ C++程序的入口是main函数（函数即方法，一个意思）

□ C++完全兼容C语言的语法，很久以前，C++叫做C with classes

小码哥教育 SEEMYGO

C++发展历史



cin、cout

- C++中常使用cin、cout进行控制台的输入、输出

```
#include <iostream>
using namespace std;

int main() {
    int age;
    cin >> age;

    cout << "age is " << age << endl;
    return 0;
}
```

- cin用的右移运算符>>，cout用的是左移运算符<<
- endl是换行的意思

小码哥教育 SEEMYGO

函数重载（Overload）

- 规则
  - 函数名相同
  - 参数个数不同、参数类型不同、参数顺序不同

- 注意
  - 返回值类型与函数重载无关
  - 调用函数时，实参的隐式类型转换可能会产生二义性

- 本质
  - 采用了name mangling或者叫name decoration技术
    - ✓ C++编译器默认会对符号名（变量名、函数名等）进行改编、修饰，有些地方翻译为“命名倾轧”
    - ✓ 重载时会生成多个不同的函数名，不同编译器（MSVC、g++）有不同的生成规则
    - ✓ 通过IDA打开【VS_Release_禁止优化】可以看到

extern “C”

- 被extern "C"修饰的代码会按照C语言的方式去编译

```
extern "C" void func() {
    cout << "func()" << endl;
}

extern "C" void func(int age) {
    cout << "func(int age) " << age << endl;
}
```

```
extern "C" {
    void func() {
        cout << "func()" << endl;
    }

    void func(int age) {
        cout << "func(int age) " << age << endl;
    }
}
```

extern “C”

- 如果函数同时有声明和实现，要让函数声明被extern "C"修饰，函数实现可以不修饰

```
extern "C" void func();
extern "C" void func(int age);

void func() {
    cout << "func()" << endl;
}

void func(int age) {
    cout << "func(int age) " << age << endl;
}
```

```
extern "C" {
    void func();
    void func(int age);
}

void func() {
    cout << "func()" << endl;
}

void func(int age) {
    cout << "func(int age) " << age << endl;
}
```

extern “C”

- 由于C、C++编译规则的不同，在C、C++混合开发时，可能会经常出现以下操作
- C++在调用C语言API时，需要使用extern "C"修饰C语言的函数声明

sum.h

```
1  #ifndef __SUM_H
2  #define __SUM_H
3
4  int sum(int a, int b);
5
6  #endif // !__SUM_H
```

sum.c

```
1  #include "sum.h"
2
3  int sum(int a, int b) {
4      return a + b;
5  }
```

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  extern "C" {
5      #include "sum.h"
6  }
7
8  int main() {
9      cout << sum(10, 20) << endl;
10     return 0;
11 }
```

extern “C”

- 有时也会在编写C语言代码中直接使用extern “C” ，这样就可以直接被C++调用

sum.h

```
1   #ifndef __SUM_H
2   #define __SUM_H
3
4   #ifdef __cplusplus
5   extern "C" {
6   #endif // __cplusplus
7
8   int sum(int a, int b);
9
10  #ifdef __cplusplus
11  }
12  #endif // __cplusplus
13
14  #endif // !__SUM_H
```

main.cpp

```
1   #include <iostream>
2   using namespace std;
3
4   #include "sum.h"
5
6   int main() {
7       cout << sum(10, 20) << endl;
8       return 0;
9   }
```

- 通过使用宏__cplusplus来区分C、C++环境

默认参数

- C++允许函数设置默认参数，在调用时可以根据情况省略实参。规则如下：
  - 默认参数只能按照右到左的顺序
  - 如果函数同时有声明、实现，默认参数只能放在函数声明中
  - 默认参数的值可以是常量、全局符号（全局变量、函数名）

```
int age = 33;

void test() {
    cout << "test()" << endl;
}

void display(int a = 11, int b = 22, int c = age, void (*func)() = test) {
    cout << "a is " << a << endl;
    cout << "b is " << b << endl;
    cout << "c is " << c << endl;
    func();
}

int main() {
    display();
    return 0;
}
```

小码哥教育 SEEMYGO

默认参数

- 函数重载、默认参数可能会产生冲突、二义性（建议优先选择使用默认参数）

```
void display(int a, int b = 20) {
    cout << "a is " << a << endl;
}

void display(int a) {
    cout << "a is " << a << endl;
}

int main() {
    display(10);
    return 0;
}
```

小码哥教育 SEEMYGO

内联函数（inline function）

- 使用inline修饰函数的声明或者实现，可以使其变成内联函数
  - 建议声明和实现都增加inline修饰

- 特点
  - 编译器会将函数调用直接展开为函数体代码
  - 可以减少函数调用的开销
  - 会增大代码体积

- 注意
  - 尽量不要内联超过10行代码的函数
  - 有些函数即使声明为inline，也不一定会被编译器内联，比如递归函数

小码哥教育 SEEMYGO

内联函数与宏

- 内联函数和宏，都可以减少函数调用的开销
- 对比宏，内联函数多了语法检测和函数特性
- 思考以下代码的区别

```
#define sum(x) (x + x)
inline int sum(int x) { return x + x; }
int a = 10; sum(a++);
```

小码哥教育 SEEMYGO

#pragma once

- 我们经常使用#ifndef、#define、#endif来防止头文件的内容被重复包含

- #pragma once可以防止整个文件的内容被重复包含

- 区别
  - #ifndef、#define、#endif受C\C++标准的支持，不受编译器的任何限制
  - 有些编译器不支持#pragma once（较老编译器不支持，如GCC 3.4版本之前），兼容性不够好
  - #ifndef、#define、#endif可以针对一个文件中的部分代码，而#pragma once只能针对整个文件

引用（Reference）

- 在C语言中，使用指针（Pointer）可以间接获取、修改某个变量的值
- 在C++中，使用引用（Reference）可以起到跟指针类似的功能

```
int age = 20;
// rage就是一个引用
int &rage = age;
```

- 注意点
  - 引用相当于是变量的别名（基本数据类型、枚举、结构体、类、指针、数组等，都可以有引用）
  - 对引用做计算，就是对引用所指向的变量做计算
  - 在定义的时候就必须初始化，一旦指向了某个变量，就不可以再改变，“从一而终”
  - 可以利用引用初始化另一个引用，相当于某个变量的多个别名
  - 不存在【引用的引用、指向引用的指针、引用数组】

- 引用存在的价值之一：比指针更安全、函数返回值可以被赋值

const

- const是常量的意思，被其修饰的变量不可修改
  - 如果修饰的是类、结构体（的指针），其成员也不可以更改

- 以下5个指针分别是什么含义？

```
int age = 10;
const int *p0 = &age;
int const *p1 = &age;
int * const p2 = &age;
const int * const p3 = &age;
int const * const p4 = &age;
```

- 上面的指针问题可以用以下结论来解决：
  - const修饰的是其右边的内容

常引用（Const Reference）

- 引用可以被const修饰，这样就无法通过引用修改数据了，可以称为常引用
  - const必须写在&符号的左边，才能算是常引用

- const引用的特点
  - 可以指向临时数据（常量、表达式、函数返回值等）
  - 可以指向不同类型的数据
  - 作为函数参数时（此规则也适用于const指针）
    - ✓ 可以接受const和非const实参（非const引用，只能接受非const实参）
    - ✓ 可以跟非const引用构成重载

- 当常引用指向了不同类型的数据时，会产生临时变量，即引用指向的并不是初始化时的那个变量

小码哥教育 SEEMYGO

数组的引用

- 常见的2种写法

```
int array[] = { 10, 20, 30 };
int (&ref1)[3] = array;
int * const &ref2 = array;
```

小码哥教育 SEEMYGO

表达式

- C++的有些表达式是可以被赋值的

```
int a = 1;
int b = 2;
// 赋值给了a
(a = b) = 3;
// 赋值给了b
(a < b ? a : b) = 4;
```

小码哥教育 SEEMYGO

引用的本质

- 引用的本质就是指针，只是编译器削弱了它的功能，所以引用就是弱化了的指针

- 一个引用占用一个指针的大小

小码哥教育 SEEMYGO

汇编语言

- 汇编课程
  - iOS底层原理（上）
  - 为什么优秀程序员都必须懂C++

- 汇编语言的种类
  - 8086汇编（16bit）
  - x86汇编（32bit）
  - x64汇编（64bit）
  - ARM汇编（嵌入式、移动设备）
  - ……

- x64汇编根据编译器的不同，有2种书写格式
  - Intel
  - AT&T

- 汇编语言不区分大小写

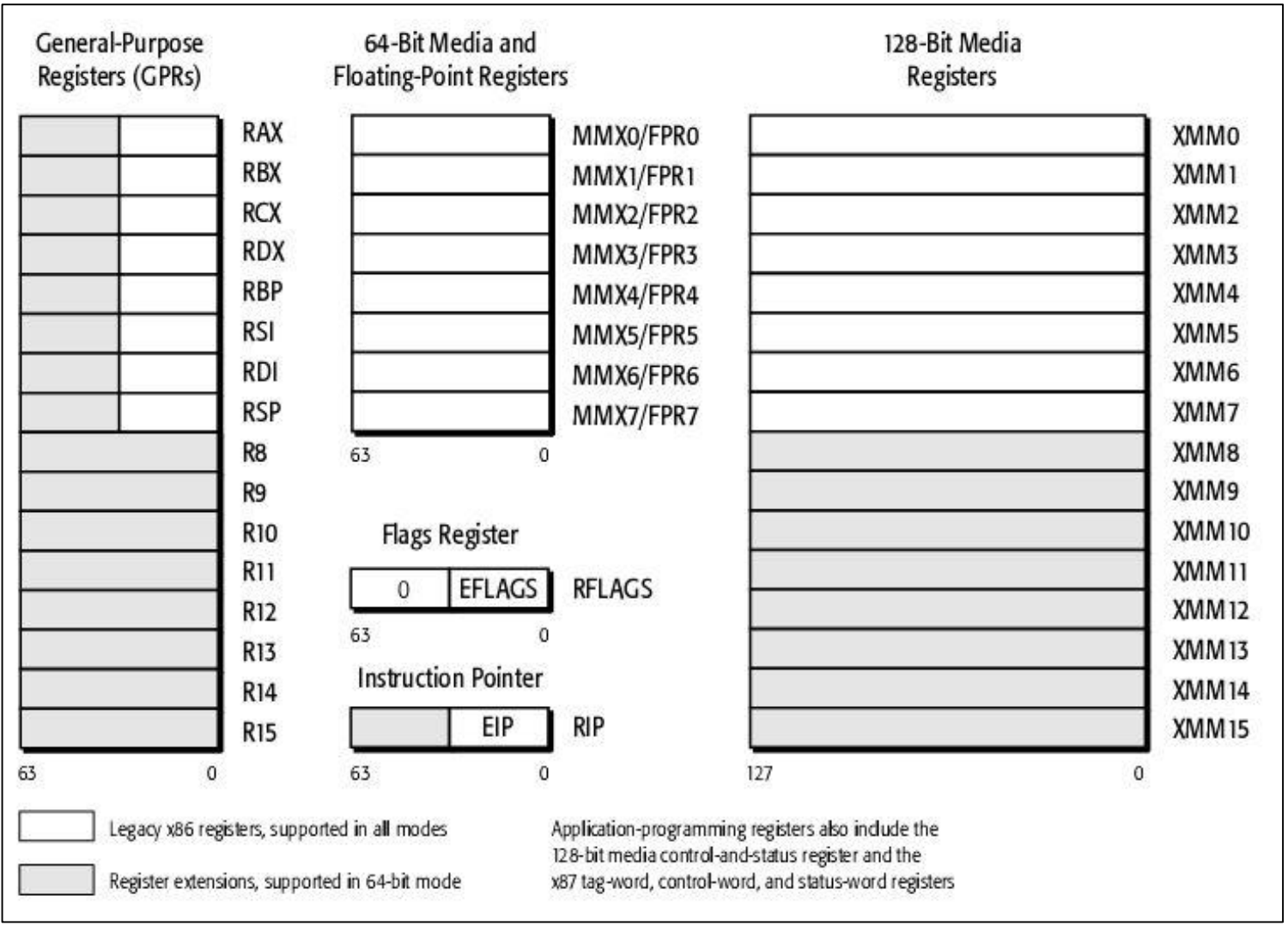
小码哥教育 SEEMYGO

AT&T汇编 vs Intel汇编

项目	AT&T	Intel	说明
寄存器命名	%eax	eax	Intel的不带%
操作数顺序	movl %eax, %edx	mov edx, eax	将eax的值赋值给edx
常数\立即数	movl $3, %eax movl $0x10, %eax	mov eax, 3 mov eax, 0x10	将3赋值给eax 将0x10赋值给eax
jmp指令	jmp *%edx jmp *0x4001002 jmp *(%eax)	jmp edx jmp 0x4001002 jmp [eax]	在AT&T的jmp地址前面要加星号*

操作数长度	movl %eax, %edx
	movb $0x10, %al
	leaw 0x10(%dx), %ax
		mov edx, eax
		mov al, 0x10
		lea ax, [dx + 0x10]
			b = byte (8-bit)
			s = short (16-bit integer or 32-bit floating point)
			w = word (16-bit)
			l = long (32-bit integer or 64-bit floating point)
			q = quad (64 bit)
			t = ten bytes (80-bit floating point)

x64汇编 – 寄存器

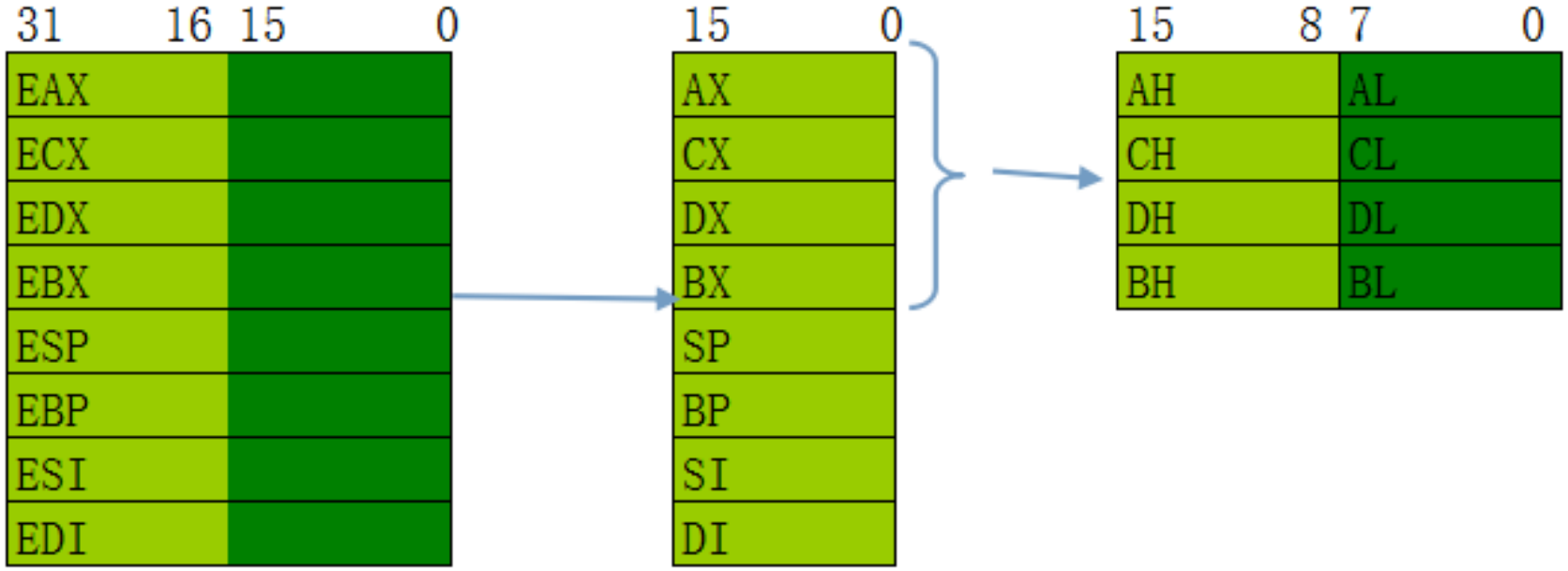



```
|63..32|31..16|15-8|7-0|
                |AH.|AL.|
                |AX.....|
          |EAX...........|
|RAX.....................|
```

- 一般的规律
  - R开头的寄存器是64bit的，占8字节
  - E开头的寄存器是32bit的，占4字节

小码哥教育 SEEMYGO

x64汇编 – 寄存器



小码哥教育 SEEMYGO

x64汇编要点总结

- mov dest, src
  - 将src的内容赋值给dest，类似于dest = src
- [地址值]
  - 中括号[ ]里面放的都是内存地址
- word是2字节，dword是4字节（double word），qword是8字节（quad word）
- call 函数地址
  - 调用函数
- lea dest, [地址值]
  - 将地址值赋值给dest，类似于dest = 地址值
- ret
  - 函数返回
- xor op1, op2
  - 将op1和op2异或的结果赋值给op1，类似于op1 = op1 ^ op2

小码哥教育 SEEMYGO

x64汇编要点总结

- add op1，op2
  - 类似于op1 = op1 + op2
- sub op1，op2
  - 类似于op1 = op1 - op2
- inc op
  - 自增，类似于op = op + 1
- dec op
  - 自减，类似于op = op – 1
- jmp 内存地址
  - 跳转到某个内存地址去执行代码
  - j开头的一般都是跳转，大多数是带条件的跳转，一般跟test、cmp等指令配合使用

- 权威参考：Intel白皮书
  - https://software.intel.com/en-us/articles/intel-sdm

小码哥教育 SEEMYGO

变量地址总结

- 一个变量的地址值，是它所有字节地址中的最小的那个地址值