用OD或者IDA进行观察，直接静态观察其实就够了；

编译过程: 转换为机器码（汇编）， int,long等类型如何转换 字符串，对象，结构，数组的长度如何判断 存和取的代码是如何编译出来的

识别代码的是编译器，把c各种对象引用转化为地址，和内存值

各种变量名在汇编中不存在，汇编只有内存和数值，这部分都是由编译器识别，将名字和内存绑定，包括类名，变量名，函数名，成员名

注意:c++传数组 传结构和对象时，是否要复制值?要栈吗？ 关于形参和实参

关于 and or xor 的二进制 特殊运算 and ffffff and 0 and自己

变量的取出 和存入都需要用到中间的寄存器 变量的中间处理计算是在寄存器完成

立即数 常量

Ptr word [偏移] 字节 双字 三字 四字

什么是变量，不设变量直接执行函数和语句会如何 ，如取地址 这叫匿名

传入一个参数 匿名指针 &a 匿名对象

结构体 数组 函数作为参数

反汇编的原理

指令集 :前缀 操作码 mode操作数类型 sib辅助mode r/m计算偏移 立即数

前缀有rep repz 跨段 fs: 转换位数 word ptr []

Sib:用来计算如[ebx+xxx]这种偏移

反汇编引擎源码

Int long short 4个字节 0~fffffff 用0补高位 内存中的多字节数据是小尾倒序的

无符号整数 都是正数 int 0~fffffff

有符号 有符号 0~7fffffff正 80000000~ffffffff负 负数补码，-绝对值 或者二进取反+1

计算机只会加法

浮点数 定点不行 整数和小数长度固定 浮点 保存小数点位置 指数域. 数据域 符号域 浮点协处理器 float4 double8字节 使用浮点寄存器

编码 IEEE float 32位 最高位是符号（0正1 负） 前8位是指数 后23位是尾数

需要科学计数法转换 12.25 - 1100.01 1.10001\*2^3

最高位:

符号：

指数:3+127 01111100这是规定 大于127为正数， 小于127为负数

尾数: 尾数在高位 低位用0补

Double 1+11+42 指数+1023

指令 浮点寄存器 栈结构 st(0)-st(7) push 先到0 如果0有的话，0压到1，新数据压到0

整型转浮点： fild会自动转换

浮点转整 call \_ftol 返回eax

开栈 sub esp,8 用于存放double

Fldi 入st(0) fst 取出st(0)到内存 [esp-4] 局部变量 fstp [esp]

fistp

作为参数

Float是以八字节方式处理，push压栈会丢失，

字符 编码ascii char unicode w\_char宽字符（双字） unicode兼容ascii ascii-gb2312

Vc中用了预编译T\_CHAR

用字符指针保存 char\* wchar\_t\* 字符串首地址

Od ida自动识别字符串 db “hello world”,

字符串 保存总长度或者保存结束符\0 ascii一个 unicode用两个字节结束

布尔 一个字节 0 或非0 任何数据类型都可以代替bool

地址和指针，引用 取地址& 常量无地址（除了const伪常量 比如100）

指针变量本身也有地址，所以可以多级指针 引用定义时就要初始化，操作指针实际上是操作

普通局部变量赋值int a=0x1000 mov dword ptr[ebp-10h],1000h 常数是立即数,int是四字节双字

定义指针和初始化地址: int \*p=&a 取地址 lea ecx,[ebp-10h] 存储到普通变量 mov dword ptr [ebp-14h],ecx 指针变量也是4字节上了双字，这里用了寄存器实现中转,不一定非ecx,eax,edx也可能

指针参数

Printf(\*p) 过程 取出p 变量中的地址 mov eax,dword ptr [ebp-14h]

从p地址读数据 mov ecx, dword ptr [eax] 然后push进去再call

指针支持的运算符 + -

数组 char arr[5]={0x01,0x23} 字符数组 初始化 要开栈吧

Mov byte ptr [ebp-14h],1 各成员赋值，立即数 关于堆栈填数字方向，每次堆栈都收缩?为什么不用push？局部变量用push吗？ 当前栈顶指针在哪?

存储首地址 int \*p= (int\*) arr 强制类型转换 取数组首地址 lea ecx,[ebp-14h]

Mov [ebp-18h],ecx

指针++ p+=1 取出p中的地址到 mov ecx,dword ptr[ebp-18h]

Add ecx,4 地址的增加值和类型有关 int 是+4 char是+1

地址值放回p mov dword ptr[ebp-18h],ecx

32位操作系统中 地址也是32位 4字节， 指针变量也是4字节

指针是局部变量或者全局变量 指针包含类型（解释信息，取地址后多少位）

引用 int a=123 mov dword ptr[ebp-4],123

Int &n=a 取a地址 lea eax,[ebp-4]

N也是一个变量,保存了a地址 mov dword ptr [ebp-8],eax

引用是依赖编译器来帮助寻址的

常量 123和”hello” 数组名 都是常量 不可修改 运行前就存在 常量数据区

在文件中就有 数据地址-基地址=文件偏移

立即数

Define 在编译阶段直接转化为常量数字

Const 仍然是变量 ，假常量，修改报错

Const int a=5 mov dword ptr [ebp-4],5

可以用指针修改数据 int \*p=&a \*p=6

可以赋值给普通变量

启动函数，真正的用户入口

Maincrestartup winmainCrtStartup 可能带w前缀

栈回溯功能，函数调用流程call stack

获取版本信息 getversion 堆空间初始化 heap\_init 多线程环境初始化 宽字符处理

获取命令行 获取环境变量 初始化全局数据c init 浮点寄存器 调用 main或者winmain

Setargv setenvp

如何识别main winmain 三个参数 四个参数 都是开头 仅有的 跟到call指令再看栈是最好的

运算

无效语句 不传递运算结果的语句

优化方法 debug和release

算术

加法 变量定义

变量加常量 b=a+1 取出a mov eax,dw ptr[] 加1 add eax,1 放回 mov [],eax

寄存器负责运算中转

常量加常量b=1+2 （直接计算结果） mov [] ,3

变量加变量(加等于操作) a=a+b 取出变量 mov ecx,[ebp-4] 加等于操作 add ecx,[]

直接加上b内存中的值 结果存入 a mov [ebp-4],ecx （也可以存到其他变量比如c=a+b）

优化的时候 会删除无用代码（没引用的变量） 重新赋值之前没有访问

合并 常量传播 编译期 变量转常量 int a=1 printf(a) 变成print(1) 如果变量没有修改过，没有取地址，间接访问，就可以当成常量

无用的赋值会被删掉

常量折叠 多个常量运算，直接折成1个 缩短cpu汇编码

动态的变量是不会常量化的

减法:sub 反码+1 等于补码 内部把减法转加法

变量减常量 取出变量到eax sub eax,1 移回变量

变量减变量 把sub eax,1改成 sub eax []

混合运算 左右顺序执行 需要一个结果寄存器

乘法 有符号 无符号 imul mul

变量乘常量（2的）取出 a\*5 mov edx,[] imul edx,edx,5

乘2的幂 a\*16 取出变量 mov eax.[] 左移代替乘法 shl eax,4

常量相乘 编译时直接算出来

混合 加法乘发呆 a\*4+2 利用lea完成？？？ Lea edx,[ecx\*4+2] （要求组合运算乘数是2,4,8）

变量相乘 取出 乘以地址指向的变量 imul ecx,[] 保存结果

变量\*常量+常量 debug 拆分 先乘后加

Release 优化 会转化成连乘 优化成lea

除法 整数（目前只讨论 ） 取余（模） 除法不保留余数

两个有符号相除才有符号 一个没符号最后结果就无符号 有符号的最高位

获得结果如何处理小数，向下取整 向上取整 向零取整

除数为整型常量时可优化，变量不行

扩展位edx是保存余数的 把eax等 高位扩展到edx ,初始值为0或ffffffff（看被除数正负）

变量除变量 需要扩展高位 cdq 取出变量到eax idiv eax,[] 此时商在eax中 可以存回变量

变量除常量 2 取变量 扩展高位 cdq 自身减去扩展高位 右移

除2的幂 a/8 取变量 cdq(看变量是正还是负，为负edx为fffff)

and edx，7 add eax.edx eax（被除数）加0或加7， 右移3

（这个and是什么，为什么+7）

7是什么

除以非2的幂 取变量 扩展cdq 常量移到寄存器? Idiv eax,ecx

取模 非2 的幂 上面一样 idiv后 edx就是余数

优化使用乘法 sub shr 公式 有符号除法

优化取模代码

算术结果溢出

无符号进位 cf 溢出of

自增自减 ++ --

自增后参与表达式

前缀b=1+(++a) 取出a到eax eax+1 存回 取出a,赋值给b

后缀 b=1+(a++) 先运算1+a 存到b 再加1

赋值

关系 > < 两边是常量还是变量

与 或 非 比较两边语句的结果 关系跳转jz je jnz js 0非0 奇偶 正负 溢出 进位 大于小于 不大于

表达式短路 ，不执行后面 的语句 a&&a=1 je

三目表达式

逻辑

位运算 左移 右移 位或 与 异或 取反

常用优化编译

设备无关 :常量折叠 常量传播x=3 y=x+3 直接y=6 减少变量

公共表达式 归并

复写传播 剪支优化 顺序代替分支

强度削弱 加法代替乘法 乘法和位移代替除法

数学变换

代码外提

目标代码：流水线 分支 高速缓存

工作流程：取指令 译码长度功能寻址方式 寻址得到操作数（寄，内，立）路径 取操作数 执行指令（cu控制，alu计算） 存放结果

流水线提前对下一个指令处理 取指令内存管理单元

指令相关性

地址相关性

分段分页页表缓冲tlb

控制语句

If if(a==0){} cmp []，0 cmp不需要取出变量,所以没用寄存器（其实用了标志）

Jne 跳转地址到结束块{}

If(a>0) cmp jle 跳到外面

If else if(a==0){} else {} cmp 然后 jne跳到else {}的首地址 if{}和else{} 代码是埃在一起的 在if和else之间有个 jmp 是执行if之后跳过else

条件表达式 三目 和if else类似

流水线优化release 建议追踪语句块还比较好 找到边界

分支优化

If else if if(a>0) {} else if(a==0) {} else {} cmp jle 跳到 第二个块 ，块之间有jmp跳出

优化：不能执行的分支可能被优化 比如常量传播

Swith 把所有的跳转放在一起 然后语句块放一起 如果不break的话会顺序执行语句

优化

1. 有序线性 跳转表
2. 间隔较大 索引表 地址表

3、判定树

Do/while /for

Do 先执行，后判断，如果成立继续执行 类似于 if goto

While 先判断，后执行，不成立跳出

For 需要一个计数器变量 赋初值 计算步长 条件比较

循环的优化 分支优化 优化成do

函数工作原理 push是压内存还是压寄存器 ret 实参复制给形参是什么事

栈帧 esp ebp 保存栈顶和栈底 esp<ebp时形成栈帧 可寻址数据 局部变量 返回地址 函数参数

一个函数 进入另一个函数时就会开辟栈空间，形成栈帧，调用结束时关闭栈帧，栈平衡

避免栈空间上溢下溢 损害其他数据

退出函数时，会对比esp和ebp 所以预先保存ebp,退出函数时还原栈底

Debug时可以用\_chkesp检测栈平衡，debug版默认退出时会调用 release时不检查，不用保存底

调用方式 cdecl 默认，调用方进行栈平衡，不定参数 stdcall被调 fastcall，寄存器传参，被调平衡

被调平衡Ret 4

主调平衡Call 函数 ; add esp,4

优化 多次调用，一次平衡:复写传播

Fastcall只用ecx和edx 其他用栈传 push4 push3 mov edx,2, mov ecx,1

传参 常数 变量

平衡栈

Ebp或者esp寻址 变量访问， int a=1 mov dword prt [ebp-4],1 ，局部变量是连续排列的 进入函数之后需要先开辟局部变量的空间 sub esp,8 函数结尾释放栈空间

Debug用ebp ebp保存esp 方便esp还原 栈平衡检查

Release中是用esp直接寻址 sub esp lea eax,[esp+] pusheax

正数标号法

负数标号法 正数是参数 负数是局变

参数

返回值 call把下一行地址压入栈 返回地址 ret时堆栈已经平衡，会修改eip,同时esp+4释放地址 返回值在eax中

返回结构体 t=getStruct()

取地址符号 b=&a lea eax,[ ] mov b,eax 成员赋值

函数流程 传参（栈或寄） call调用，返回地址压栈 保存栈底ebp 申请栈空间sub esp 保存寄存器环境（将被用来中转的寄存器） 函数语句 还原寄存器 平衡局部变量的栈 ret返回 平衡栈顶

变量位置和访问方式 作用域 生命周期 {} 全局属于进程 静态属于文件 局部属于函数

全局变量 常量不可写 全局和静态可读写 初始值会被写到pe中 第一条代码前就已经分配内存了，全局变量不用栈指针 直接内存访问[]，有一个固定的地址值

多个全局的顺序先低后高，连续

全局静态 只是受编译检查限制不能在文件外用，可以指针直接改

局部变量

局部静态变量 不会随着作用域而结束 ，生命周期和全局相同，没进入作用域就已经存在，

静态变量有个标志，表示是否初始化过， 如果初始化过，再定义无效

名称粉碎法，编译期重新命名

堆变量 malloc new 返回申请的堆地址 free delete需要空间首地址（不能丢失）

保存首地址的指针

需要执行函数 找到new free等

堆空间 有个结构 CrtMemoryBlockHeader 堆数据的尾部 越界检查标志

数组和指针寻址

函数内 局部变量 int a[5]={1,2,3,4,5} 占用20字节

赋值 mov [],1 数组连续，类型一致

字符串本身就是数组，赋值是每次复制4字节比较快

Release下字符串处理函数 是内联函数 不会call,会嵌入代码

作为参数 首地址作为指针传递

作为返回值

下标寻址，指针寻址 a[1] p+1 指针属于变量，需要先取地址， 再算偏移

数组下标可以直接读内存

1、下标是常数 首地址 [ebp-8] a[2]通过首地址直接算出偏移 [ebp-6]

2、下标是变量a[i] [ebp-8+ecx\*4]

3、下标是表达式 先算表达式结果 ，常量会折叠，变量会动态

多维数组

数组中的指针

指向数组的指针

函数指针

结构体和类 都有构造函数 析构 成员函数 只有public和private区别，没区别

对象大小只包含数据，成员函数属于执行代码，不属于对象数据

对象内存布局 内存对齐不一定连续排列 虚函数表 父类数据成员

This 指针 保存了所属对象的首地址 p->member 访问成员->和成员名，都是由编译器识别

成员访问 取对象首地址

默认约定时会用ecx保存对象首地址（this） 并把ecx传入到 内部 push ecx

然后调用函数时用到了成员变量就会自动加上this指针访问

Stdcall用栈传递，

调用成员函数（定义时this->成员可以省略，写成成员）

传参过程

函数的初始化过程

静态成员，和静态局部变量原理一样，直接寻址，难以和全局变量区分

对象作为参数 如何传入对象数据的? 传入的是 重新复制了对象?

对象作为返回值

构造函数，析构函数

出现时机

有构造吗？

拷贝构造函数

何时调用构造?

局部对象

堆对象

参数对象

返回对象

全局对象 所有全局对象在同一地点调用构造 cinit() 定位初始化 栈回溯

静态对象

默认构造函数 什么都不做 可能直接被优化掉 （除了有虚函数 ，成员对象有构造函数）

虚函数 虚表指针 在对象首地址 前4字节，有默认构造函数 虚函数首地址 存在虚函数地址表， 会添加虚表指针这个隐藏的数据成员，总大小会+4

默认构造（编译器自动插入代码） 获取虚表首地址，赋给指针 初始化

访问虚函数时，根据对象首地址取虚表，获取虚函数首地址，调用

对象调用自身虚函数不需要用虚表，只有多态的时候

机制

识别

调用识别

继承和多重继承

类与类的关系

多重继承

虚基类（抽象类） 纯虚函数，没有实现

异常处理 try throw catch 异常回调 异常抛出

函数入口 设置回调 eax=FuncInfo CxxFrameHandler() 抛出信息 回调中异常对象的地址，ThrowInfo数据地址，FuncInfo表结构地址 由异常类型，try块匹配

没找到try块

找到try块 根据表 找catch信息表 找catch块

栈展开，产生异常对象

析构已结束对象

执行catch代码

异常类型是基本数据类型

异常类型是对象

识别