

Activity

.idb Files
(SPARC V8+
Unix 32位 **ELF Files**
经过IDA初步解析的
数据库文件)

C Files
(*h、*.c)

解析生成

汇编代码

汇编->C

Main
Work



```
.text:0001256C ? int __cdecl main(int argc, const char **argv, const char **envp)
.text:0001256C .global main
.text:0001256C main:
.text:0001256C var_4 = -4
.text:0001256C
.text:0001256C save %sp, -0x68, %sp
.text:00012570 sethi %hi(loc_22C00), %17
.text:00012574 call sub_12330
.text:00012578 set loc_22F94, %17
.text:0001257C clr [%fp+var_4]
.text:00012580 mov 0x30, %o0
.text:00012584 set 0x62C, %g1
.text:0001258C ld [%17+g1], %l0
.text:00012590 call _PLXAmem_malloc
.text:00012594 mov %l0, %o1
.text:00012598 ld [%l0], %g1
.text:0001259C cmp %g1, 0
.text:000125A0 bne %icc, loc_125CC
.text:000125A4 sethi %hi(0x400), %g1
.text:000125A8 set 0x4C, %g1
.text:000125B0 ld [%17+g1], %o0 ! env
.text:000125B4 set 0x3C, %g1
.text:000125B8 ld [%17+g1], %o1 ! char *
.text:000125C0 call __assert
.text:000125C4 mov 0xAF, %o2
.text:000125C8 sethi %hi(0x400), %g1
.text:000125CC loc_125CC:
.text:000125CC set 0x62C, %g1 ! CODE XREF: main+34↑j
```

目录

- 准备知识
- SPARC汇编语言
- IDA工具的使用
- 从汇编到C
- 例子
- 学习资料

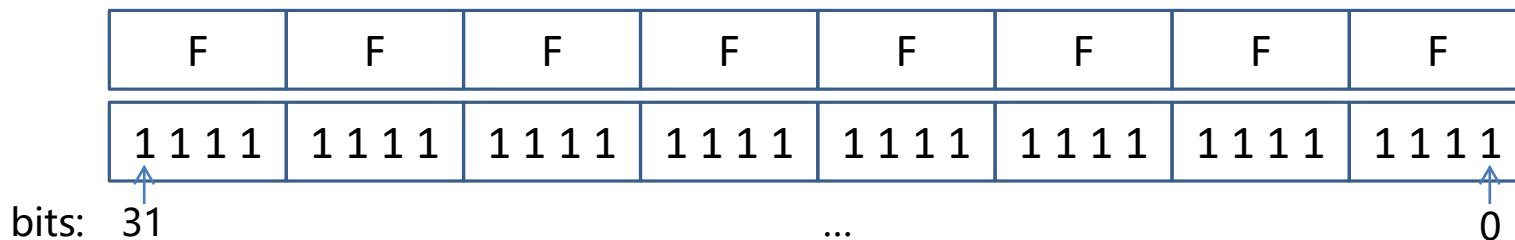
一、准备知识

- 二进制、十六进制

decimal	hex	binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

- 二进制：0、1，一位占用1个bit。
- 十六进制：0-9、A-F，一位占用4个bit，以0x开头表示。

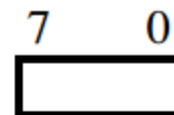
$-1 = 0xFFFFFFFF = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111$ (binary)



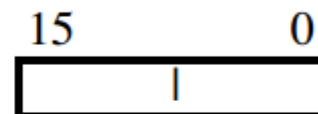
一、准备知识

- 准备知识-数据大小

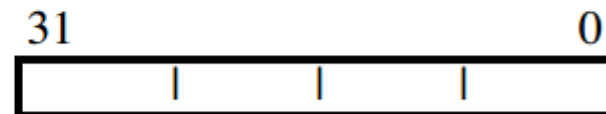
(unsigned) char byte:



(unsigned) short half (2 bytes):

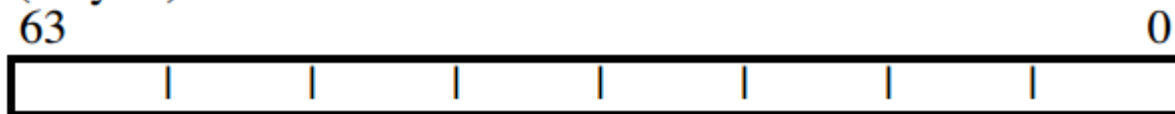


(unsigned) int、long
enum、float、pointer word (4 bytes):



double

double (8 bytes):



一、准备知识

- 练习

```
typedef enum  
{  
    A,  
    B,  
} enum1;
```

sizeof(c) = **1**

sizeof(ttt) = **100**

```
void Func(char str[100])  
{  
    char c = 's';  
    char ttt[100] = "Hello";  
    char *p = ttt;  
    void *pv = malloc(100);  
    enum1 e = A;  
    double d = 100.0;  
    ...  
}
```

sizeof(p) = **4**

sizeof(pv) = **4**

sizeof(e) = **4**

sizeof(d) = **8**

sizeof(str) = **4**

一、准备知识

- **数据的存储地址**

数据存储在 “基地址+偏移地址” 的内存单元中。

结构体内或数组内的数据元素以地址递增的顺序进行存储。

[Address] : 表示对起始地址为Address的内存单元进行访问。

ELEMENT e[100];

假设第一个元素存储的起始地址为: BASE_ADDR-0x100;

那么第i个元素存储的起始地址为:

BASE_ADDR-0x100+i*sizeof(ELEMENT);

在结构体中，通常以各元素相对于结构体变量首地址的偏移量来标识数据元素的存储地址(起始地址)。

typedef struct

```
{  
    int a[2];    //start_address = 0x0  
    double d[100]; //start_address = 0x8  
}
```

一、准备知识

- **数据对齐**

数据在内存存储时必须按其类型“对齐”。

一个short存储在内存中的起始地址必须被“2”整除；

一个int、enum存储在内存中的起始地址必须被“4”整除；

一个double存储在内存中的起始地址必须被“8”整除；

含有“double”的结构体变量的起始地址及其大小必须被“8”整除。

```
typedef struct
{
    int a;           //start_address: 0x0
    double b;        //start_address: 0x8
} struct_1;         //size = 0x10

typedef struct
{
    int a;           //start_address: 0x0
    struct_1 s1;      //start_address: 0x8
} struct_2;         //size = 0x18
```

一、准备知识

- 练习

```
typedef union
{
    double d;
    char c;
} union1;
```

```
typedef struct
{
    enum1 e;
    union1 u; //start_address = ?
    int i;    //start_address = ?
} struct1;  //size = ?
```

0x18

0x8

0x10

一、准备知识

- 练习

```
typedef struct
{
    int a;
    double b;
} struct1;
```

```
typedef struct
{
    int nr;
    struct1 pData[0x100];
} struct2;
```

```
typedef struct
{
    int nr;
    struct1 *pData;
} struct3;
```

```
struct2 s2;
struct3 s3;
s3.pData = malloc(0x100);
```

s2.pData[i].b的存储地址 = ? **(char *) &s2 + 0x8 + 0x10 * i + 0x8**

s3.pData[i].b的存储地址 = ? **[(char *) &s3 + 0x4] + 0x10 * i + 0x8**

一、准备知识

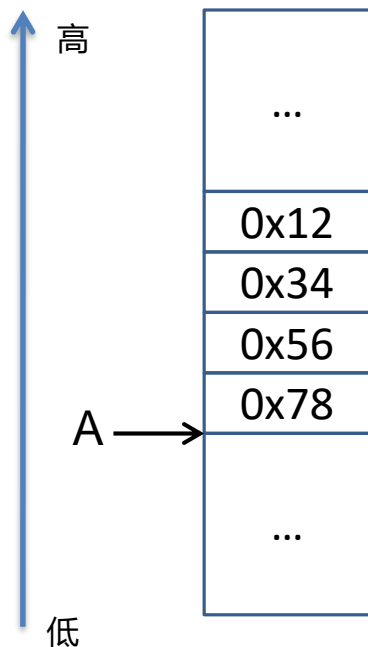
- 大小端

在计算机中以字节为单位的进行处理数据，每个地址单元都对应着一个字节，一个字节为8bit。以0x12345678为例：

[A] = 0x12345678;

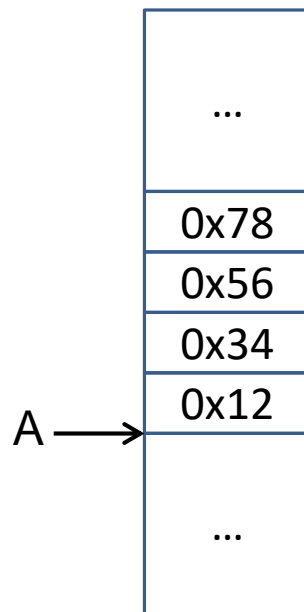
- 小端Little-Endian：

低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。



- 大端Big-Endian:

高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。



Tips : SPARC架构采用大端模式

二、SPARC汇编语言V8+

• 指令结构

$op \quad reg_{rs1}, reg_or_imm, reg_{rd}$

- 一条指令，有三个操作数(32位)，操作数有两种：寄存器和立即数。
- 伪指令：有些指令如cmp, jmp等，其操作数只有1个或者2个，通过查指令表可以得到其三操作数等价指令。
- [add]：表示对地址add所存放的内容进行操作。

指令举例

add %l2, %l3, %g3	! 本地寄存器%l2加上%l3,结果存入到全局寄存器%g3中
ld [%g3], %fsr	! 读取g3地址的存储单元内容，放入到浮点状态寄存器fsr中
st %fsr, %o2	! 把fsr中的内容，存到输出寄存器%o2中
subcc %g0, %o2, %g0	! %g0和%o2的减法运算，并对状态寄存器的条件码作相应操作
bne ,a s1	! 分支指令，如果%o2不为零，则跳到标识符为s1处

二、SPARC汇编语言V8+

- 寻址方式（1）

寻址方式是根据指令编码中给出的地址码字段来寻找真实操作数的方式。

- 立即寻址

- 操作数是直接通过指令给出，数据就包含在指令的32位编码中

`and %o2, 0x0f, %o3` ! $\%o3 \leftarrow \%o2 \ \& \ 0x0f$

- 寄存器寻址

- 寄存器中的数值作为操作数，指令中地址码给出的是寄存器编号

`add %l0, %l1, %l2` ! $\%l2 \leftarrow \%l0 + \%l1$

- 寄存器间接寻址

- 寄存器用 “[]” 括起来，表示内容，寄存器的值为某个存储器地址，操作数则放在该地址单元中。

`ld [%o2], %f0` ! $\%f0 \leftarrow \text{mem}[\%o2]$

二、SPARC汇编语言V8+

- 寻址方式 (2)

- 基址加偏址寻址

- 也称为变址寻址，就是将基址寄存器的内容与指令中给出的偏移量（偏移地址）相加，形成存储器的有效地址，用于访问基址附近的存储器单元。

`ld [%g1 + 96], %o1 ! %o1 ← mem[%g1 + 96]`

偏移地址也可以是另一个寄存器。

`ld [%o0 + %i0], %o2 ! %o2 ← mem[%o0 + %i0]`

- 相对寻址

- 基地址为程序计数器PC的变址寻址，偏移量指出了目的地址与现行指令之间的相对位置，偏移量与PC提供的基地址相加后得到有效的目的地址。

`add %g1, %g2, %g1`

`ba SUM3 !转移到SUM3`

二、SPARC汇编语言V8+

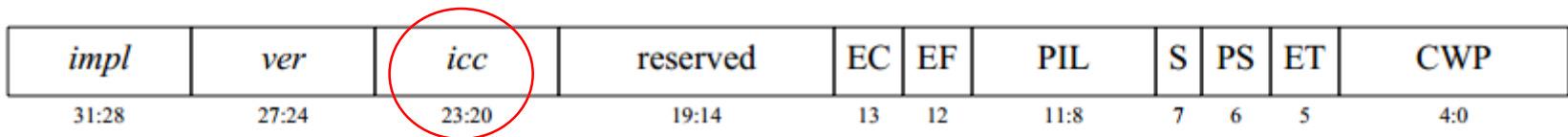
- 寄存器

寄存器	位宽
整型寄存器	32bits
浮点寄存器	32bits
状态寄存器	32bits

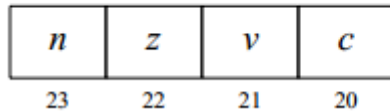
寄存器是以 “%” 开头的，如%i1、%sp、%o6等均表示寄存器。

二、SPARC汇编语言V8+

- 状态寄存器 (Processor State Register)



Integer Condition Codes (icc) Fields of the PSR



- n: negative
- z: zero
- v: overflow
- c: carry

所有带有cc后缀的指令，都会改变%icc

%xcc：扩展的整形条件码，用于64位扩展

二、SPARC汇编语言V8+

%icc, 寄存器通常用于跳转指令

```
cmp    %i1, 0
bne    %icc, loc_2A14
mov    %i1, %o0
```

BNE	1001	Branch on Not Equal	not Z
BE	0001	Branch on Equal	Z
BG	1010	Branch on Greater	not (Z or (N xor V))
BLE	0010	Branch on Less or Equal	Z or (N xor V)
BGE	1011	Branch on Greater or Equal	not (N xor V)
BL	0011	Branch on Less	N xor V
BGU	1100	Branch on Greater Unsigned	not (C or Z)
BLEU	0100	Branch on Less or Equal Unsigned	(C or Z)
BCC	1101	Branch on Carry Clear (Greater than or Equal, Unsigned)	not C
BCS	0101	Branch on Carry Set (Less than, Unsigned)	C
BPOS	1110	Branch on Positive	not N
BNEG	0110	Branch on Negative	N
BVC	1111	Branch on Overflow Clear	not V
BVS	0111	Branch on Overflow Set	V

二、SPARC汇编语言V8+

- 浮点型寄存器 (Floating-Point Registers)



- 浮点型寄存器也是32个，一般在操作浮点数的时候会用到。
- 操作其的指令通常以 “d” 为后缀;
- 通常用连续的两个浮点型寄存器来存储double。

`ldd [%g2], %f8`

`!%f8 = [%g2]`

`!%f9 = [%g2 + 4]`

`std %f8, [%fp+var_90]`

二、SPARC汇编语言V8+

- 整型寄存器 (Integer Registers)

	Register	Synonyms	Usage		
global	%g0	%r0	Always discards writes and returns zero First of seven registers for data with global context	%g0 永远为0	
	%g1	%r1			
	%g2	%r2			
	%g3	%r3			
	%g4	%r4			
	%g5	%r5			
	%g6	%r6			
	%g7	%r7			
out	%o0	%r8	First of six registers for local data and arguments to called subroutines	in和out用于给函数传递参数用	
	%o1	%r9			
	%o2	%r10			
	%o3	%r11			
	%o4	%r12	Stack pointer Called subroutine return address		
	%o5	%r13			
	%sp	%r14 %o6			
	%o7	%r15			
local	%l0	%r16	First of eight registers for local variables	%o6为栈指针，同%sp %i6为帧指针，同%fp	
	%l1	%r17			
	%l2	%r18			
	%l3	%r19			
	%l4	%r20			
	%l5	%r21			
	%l6	%r22			
	%l7	%r23			
in	%i0	%r24	First of six registers for incoming subroutine arguments		
	%i1	%r25			
	%i2	%r26			
	%i3	%r27			
	%i4	%r28			
	%i5	%r29			
	%fp	%r30 %i6	Frame pointer Subroutine return address		
	%i7	%r31			

二、SPARC汇编语言V8+

• 函数栈帧 (stack frame)

定义：

函数调用时栈中分配的内存块。

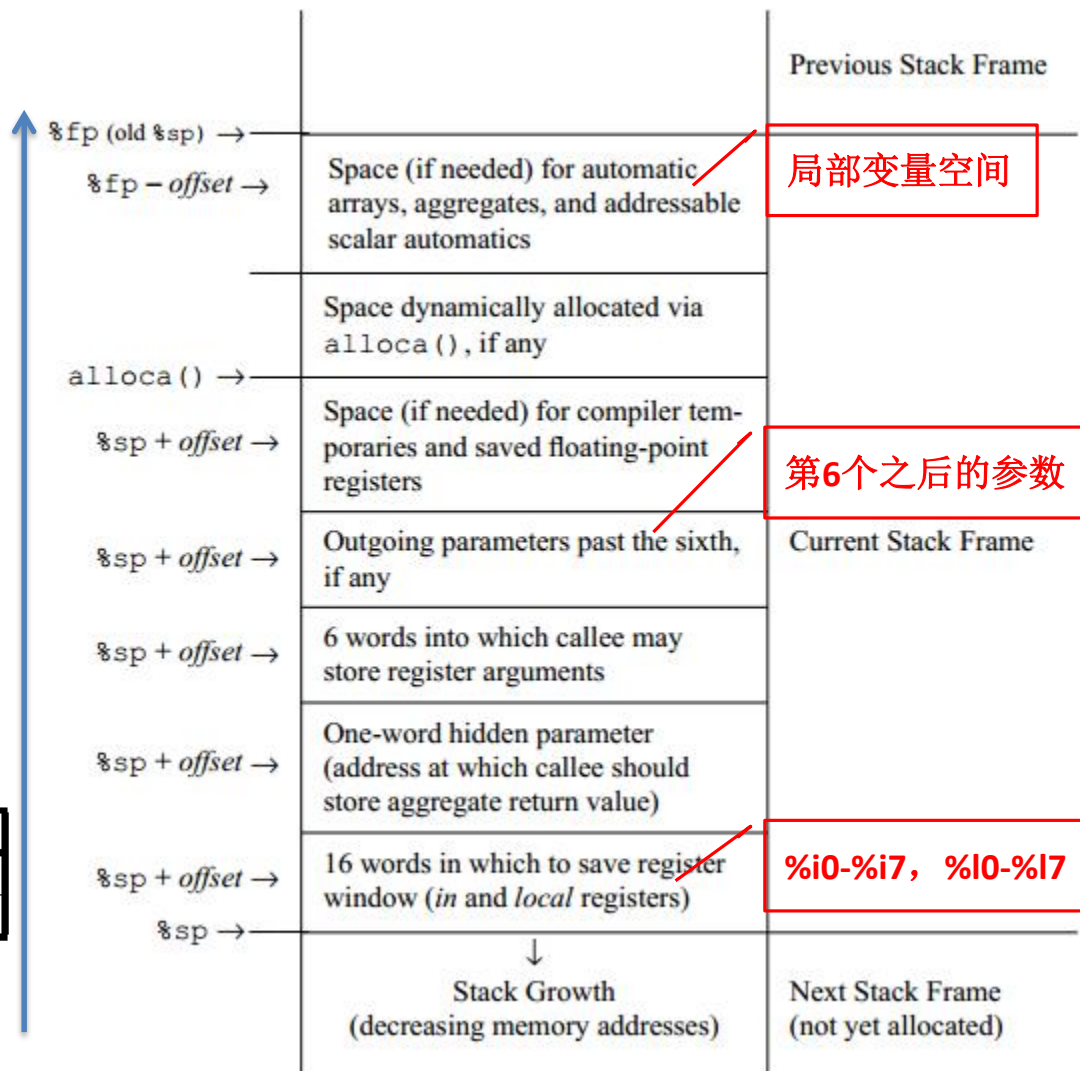
%fp：帧指针

%sp：栈指针

用途：

1. 存储调用方传入的函数参数；
2. 存储函数内局部变量。

Opcode	op3	Operation
SAVE	11 1100	Save caller's window
RESTORE	11 1101	Restore caller's window



二、SPARC汇编语言V8+

- 函数参数传递 (1)

```
A()  
{  
    ....  
    B(arguments list);  
    ....  
}
```

程序执行到函数A时（尚未调用B函数之前），将函数参数依次存入到：

%o0, %o1, %o2, %o3, %o4, %o5 ,

%sp+0x5C, %sp+0x60, %sp+0x64 ...

如果是double型参数，则使用连续的两个寄存器。

二、SPARC汇编语言V8+

- 函数参数传递（2）

```
B ( . . . )  
{  
    . . .  
}
```

运行到调用B函数的语句时，程序进入B函数中，在B函数中，程序只要对应地读取*%i0*, *%i1*, *%i2*, *%i3*, *%i4*, *%i5*, *%fp+0x5C*, *%fp+0x60*, *%fp+0x64* ... 即可获取到参数。

在B程序的最后，设置*%i0 (%f0)* 寄存器，作为**返回值**。

返回A中继续执行时，读取*%o0 (%f0)* ，即可得到B函数执行后的**返回值**。

二、SPARC汇编语言V8+

• 分支指令延时间隙

函数调用(call)、条件跳转 (ba, ble, bne,...)

- 分支语句执行前，紧跟分支指令后面的指令先被执行。
- 在跳转指令的延时间隙里不方便放有用的指令的情形时，SPARC提供了“nop”复合指令。nop的执行，它不改变任何寄存器或内存的值。
- 用加后缀“,a”来声明跳转间隙无效。如果**条件分支指令没有发生（不跳转）**，则跳转间隙中的**指令无效（不执行）**。如 bg ,a top。

$ba\{, a\}\{, pt , pn\}$	$i_or_x_cc, label$
$bn\{, a\}\{, pt , pn\}$	$i_or_x_cc, label$
$bne\{, a\}\{, pt , pn\}$	$i_or_x_cc, label$
$be\{, a\}\{, pt , pn\}$	$i_or_x_cc, label$
$bg\{, a\}\{, pt , pn\}$	$i_or_x_cc, label$
$ble\{, a\}\{, pt , pn\}$	$i_or_x_cc, label$
$bge\{, a\}\{, pt , pn\}$	$i_or_x_cc, label$
$bl\{, a\}\{, pt , pn\}$	$i_or_x_cc, label$

二、SPARC汇编语言V8+

• 练习

对于函数：

```
bool my_max(double d1, double d2, double d3, double d4);
```

对应SPARC汇编代码中的函数参数，如下解释正确的是：

- A** %i0、%i1组成d1,%i2、%i3组成d2,%i4、%i5组成d3,%fp+0x5C、%fp+0x60组成d4;
- B** %i0、%i1组成d1,%i2、%i3组成d2,%i4、%i5组成d3,%fp-0x5C、%fp-0x60组成d4;
- C** %i0、%i1组成d1,%i2、%i3组成d2,%i4、%i5组成d3,%sp+0x5C、%sp+0x60组成d4;
- D** %i0=d1,%i1=d2,%i2=d3,%i3=d4;

A

二、SPARC汇编语言V8+

• 练习

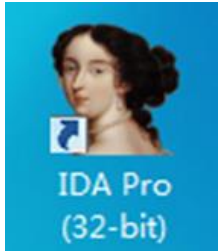
对于如下汇编代码，解释正确的是：

```
cmp      %g1, 0  
bne,a    %icc, loc_1DB60  
inc      %l1  
sethi    %hi(0), %g1
```

- A 当%g1 == 0, 指令“inc %l1”会在跳转前执行；
- B 当%g1 == 0, 指令“inc %l1”不会执行；
- C 当%g1 != 0, 指令“inc %l1”会在跳转前执行；
- D 当%g1 != 0, 指令“inc %l1”不会执行；

B、C

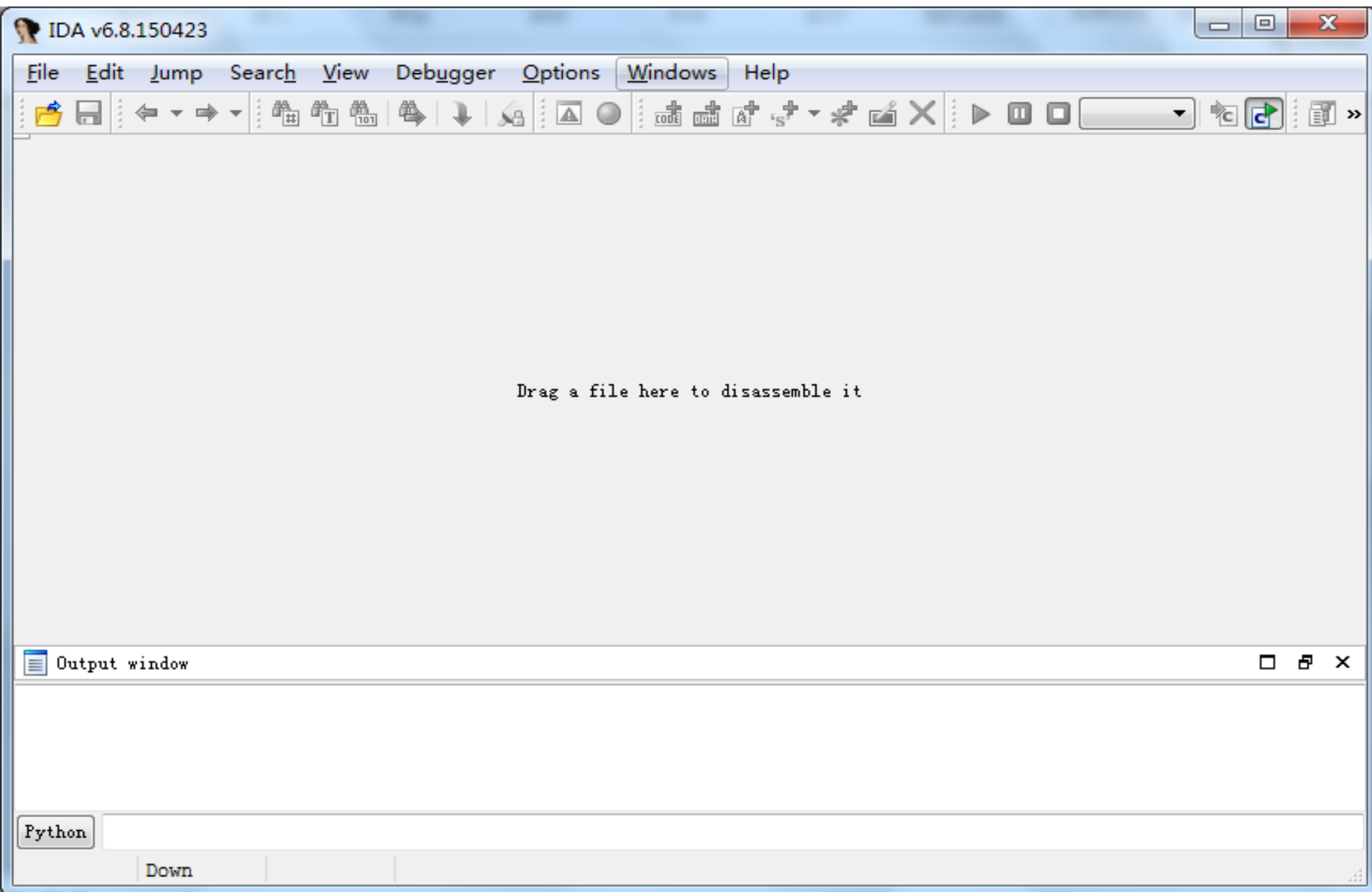
三、IDA 工具的使用



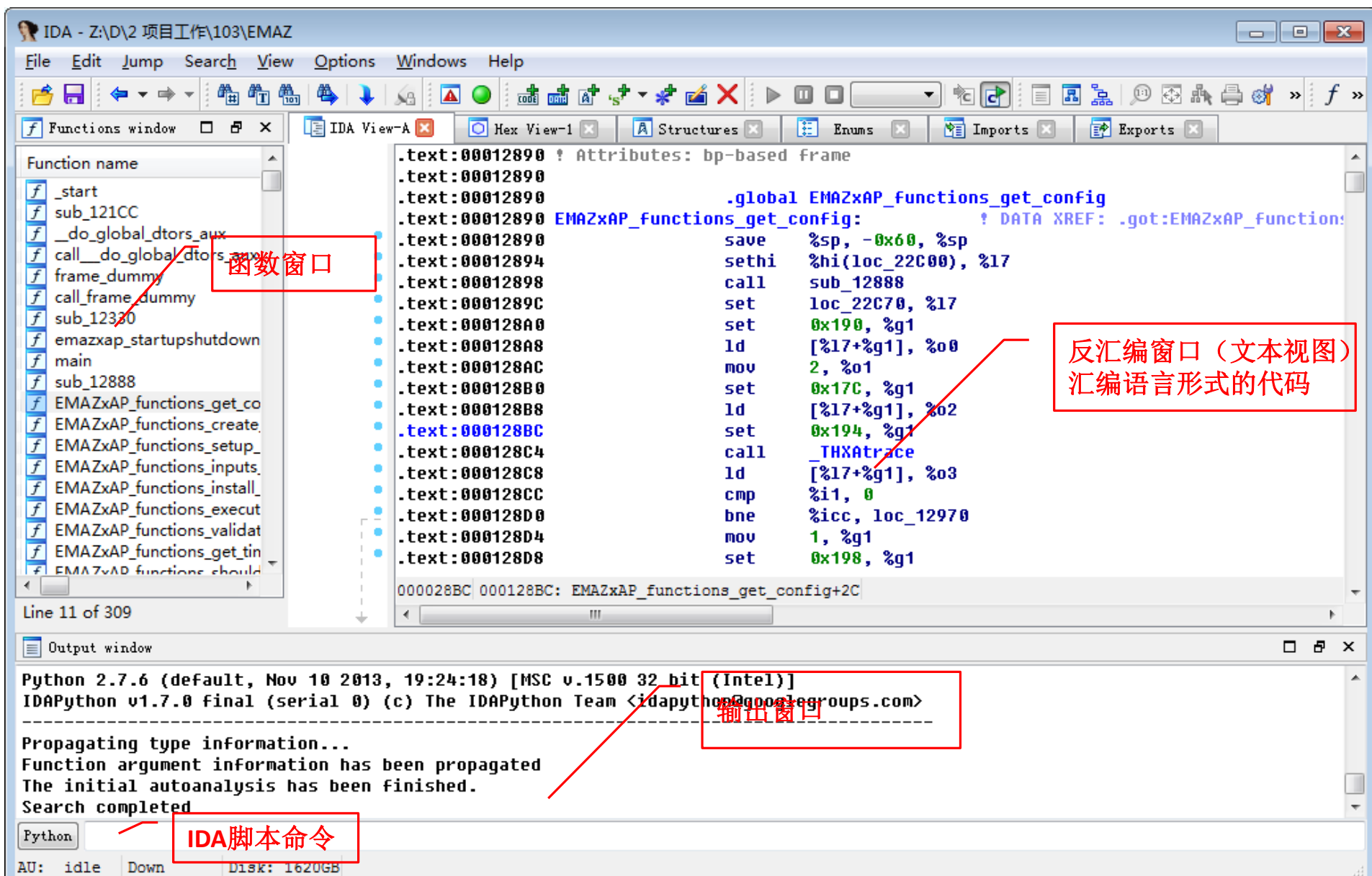
- 交互式反汇编器专业版
(**I**nteractive **D**isassembler **P**rofessional)
- 比利时列日市 (Liège) 的Hex-Rays公司的一款产品
- 版本：**IDA Pro 6.8 32-bit**



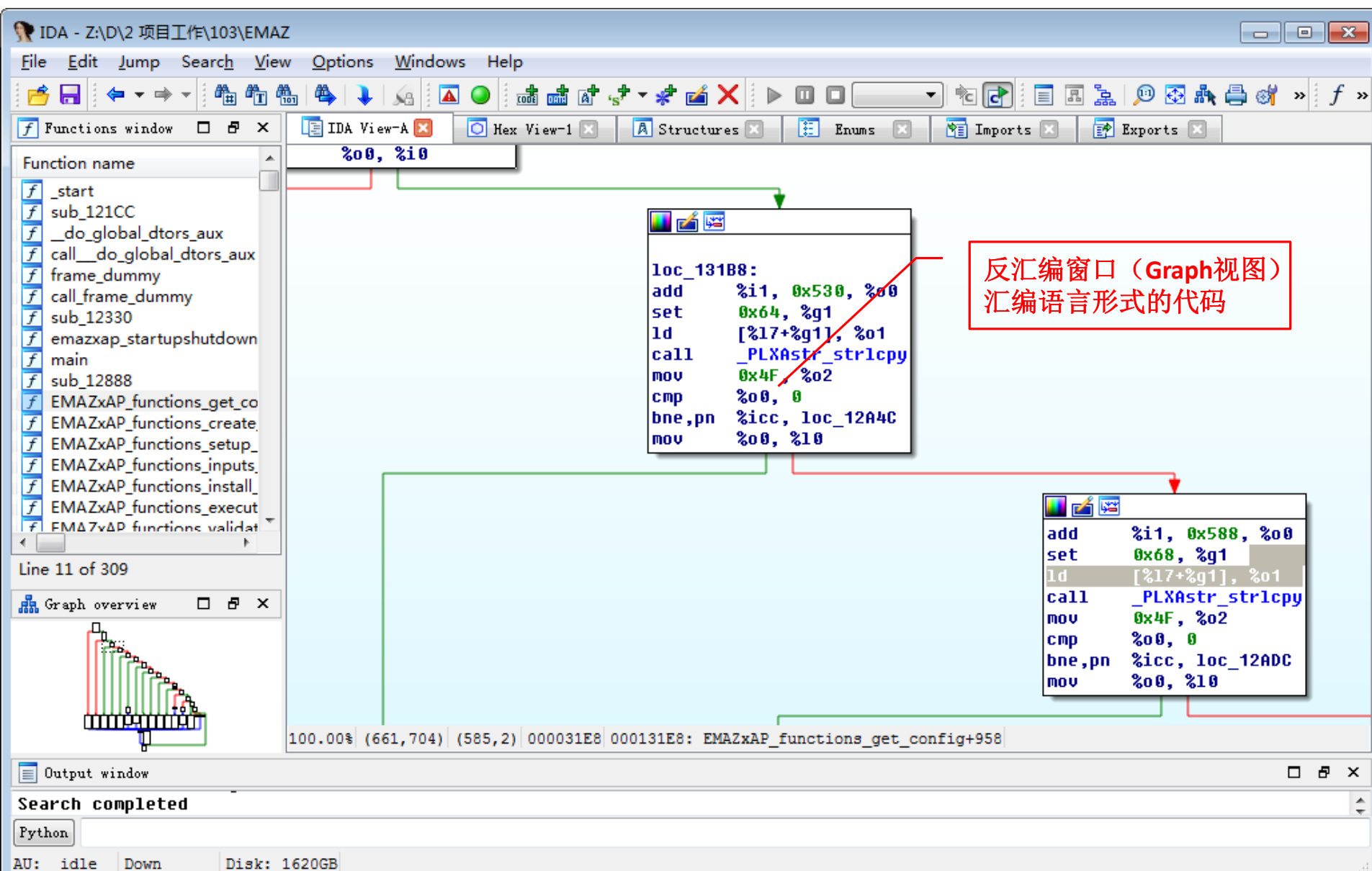
三、IDA 工具的使用



三、IDA 工具的使用



三、IDA 工具的使用



三、IDA 工具的使用

IDA - Z:\D\2 项目工作\103\EMAZ

File Edit Jump Search View Options Windows Help

Functions window

Function name

- f _start
- f sub_121CC
- f __do_global_dtors_aux
- f call__do_global_dtors_aux
- f frame_dummy
- f call_frame_dummy
- f sub_12330
- f emazxap_startupshutdown
- f main
- f sub_12888
- f EMAXxAP_functions_get_co
- f EMAXxAP_functions_create
- f EMAXxAP_functions_setup
- f EMAXxAP_functions_inputs
- f EMAXxAP_functions_install
- f EMAXxAP_functions_execut
- f EMAXxAP_functions_validat
- f EMAXxAP_functions_get_tin
- f EMAXxAP_functions_should

Address Ordinal Name

00076F80		EMMIXAxMAT_create_equation
00076F84		ERXAsignalUnInstall
00076F88		pthread_once
00076F8C		EMMIXAxFIL_resample_interpolate
00076F90		PLXAtask_suspend
00076F94		PLXAmem_malloc
00076F98		atexit
00076F9C		PLXAmem_calloc
00076FA0		strncpy
00076FA4		CNXAtaskInit
00076FA8		GTMWXTxLS_measure_height
00076FAC		THXAcheckSimMode
00076FB0		PLXAstr_strlcat
00076FB4		TLXTxGD_add_xy_multi_plot
00076FB8		ERXA_nr_sigsegu
00076FBC		GTMWXTxCONST_MIXAxMC_load
00076FC0		TLXTxDS_alloc
00076FC4		GTMWXTxWS_move_chuck_to_measure_position
00076FC8		abort
00076ECC		ERXAsignalInstall

Library

导入窗口
外部文件或C标准库
定义的函数/变量。

Line 11 of 309

Line 1 of 95

Output window

Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)]
IDAPython v1.7.0 Final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
Search completed

Python

AU: idle Down Disk: 1620GB

三、IDA 工具的使用

The screenshot shows the IDA Pro interface with the 'Exports' window open. A red callout box points to the 'Exports' window with the following text:

导出窗口
导出给其他文件使用的函数/变量。

The 'Exports' window displays a list of exported functions and their addresses. The list is as follows:

Name	Address	Ordinal
EMAZxAP_functions_create_data_structure	0001339C	
EMAZxAP_functions_destroy_data_structure	0001660C	
EMAZxAP_functions_execute_step_and_update_results	000144F4	
EMAZxAP_prepare_second_chunk	0001C984	
EMAZxAP_MCs_set_mirror_map_to_MC_value	00018F0C	
EMAZxAP_functions_setup_data_structure	00013A44	
EMAZxAP_measure_cycle	0001B238	
_PROCEDURE_LINKAGE_TABLE_	00035B58	
EMAZxAP_functions_should_step_be_executed	000141A4	
EMAZxAP_calc_measure_cycle_results	00016AC8	
EMAZxAP_MCs_ReadMCs	00018FD0	
EMAZxAP_MCs_set_mirror_map_temporary_to_zero	00018CB8	
.term_proc	00023228	
EMAZxAP_Graphics_create_files	000183F8	
EMAZxAP_functions_clear_result_parameters	0001624C	
EMAZxAP_TL_store_test_log	0001D93C	
EMAZxAP_TL_retrieve_test_log	0001D578	
EMAZxAP_prepare_finish_measurements	0001CC90	
EMAZxAP_MCs_SaveMCs	0001A2D4	
EMAZxAP_MCs_CalcMCs	000194FC	

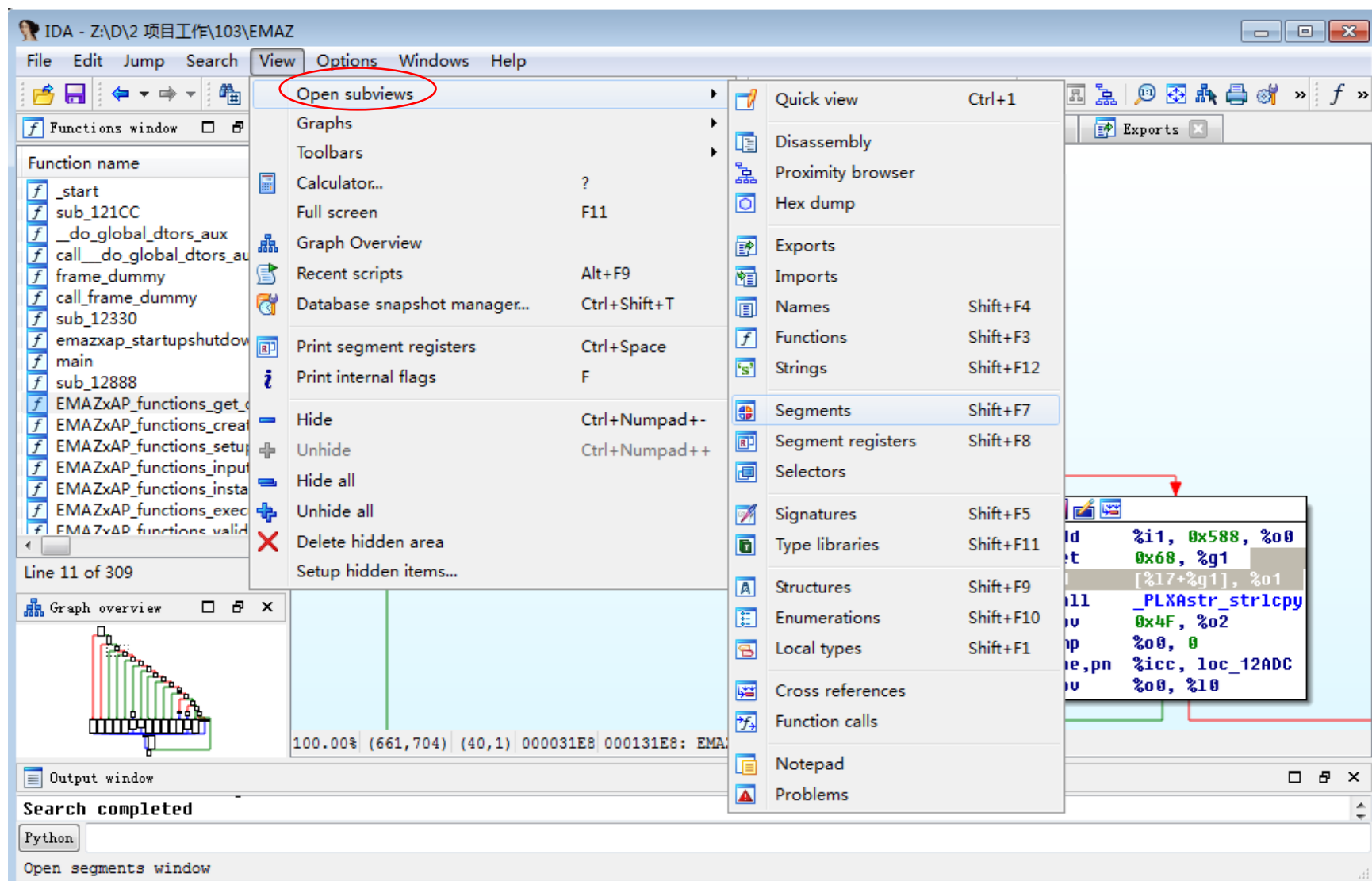
The bottom of the screenshot shows the 'Output window' with the following text:

```
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)]
IDAPython v1.7.0 Final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
-----
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
Search completed
```

At the bottom, there is a status bar showing 'AU: idle', 'Down', and 'Disk: 1620GB'.

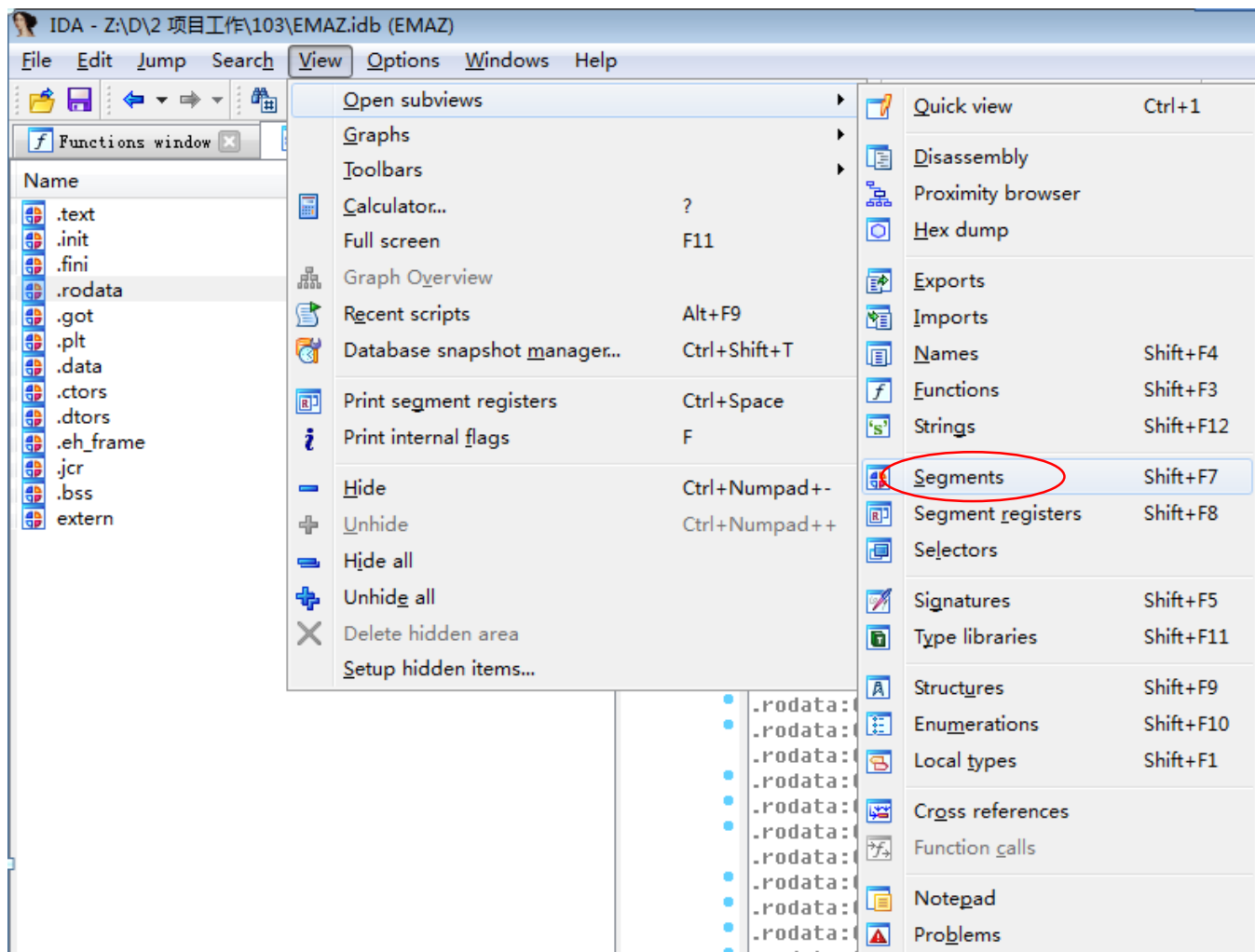
三、IDA 工具的使用

- 打开其他视图



三、IDA 工具的使用

• 段视图



.text: 代码段

.rodata: 只读数据段

.data: 全局数据段

.bss: 全局数据段

(未初始化)

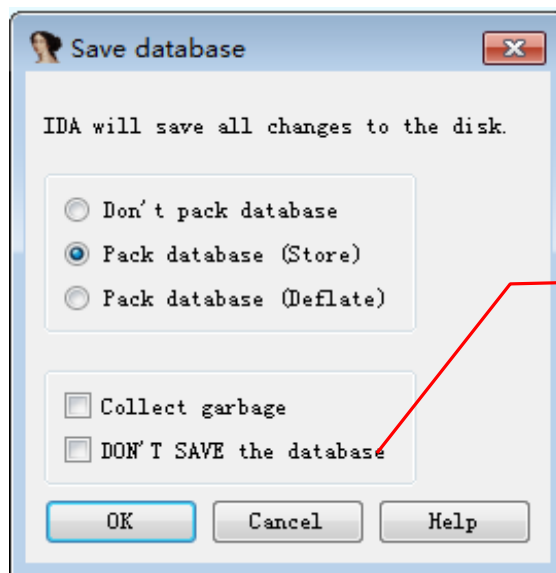
.extern: 外部符号段

.got: 全局偏移段

三、IDA 工具的使用

• IDA数据库文件

- 每次打开一个文件，同步生成.id0、.id1、.nam、.til四个数据库文件。
- 关闭IDA时，可将数据库文件打包存为一个.idb文件，这个包含了解析的全部信息，下次打开文件，可直接打开.idb文件。

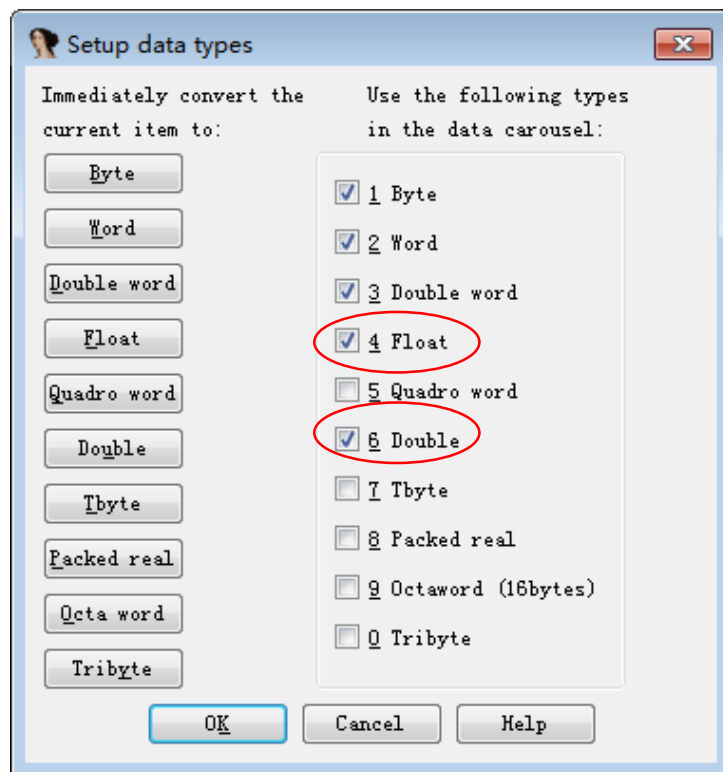
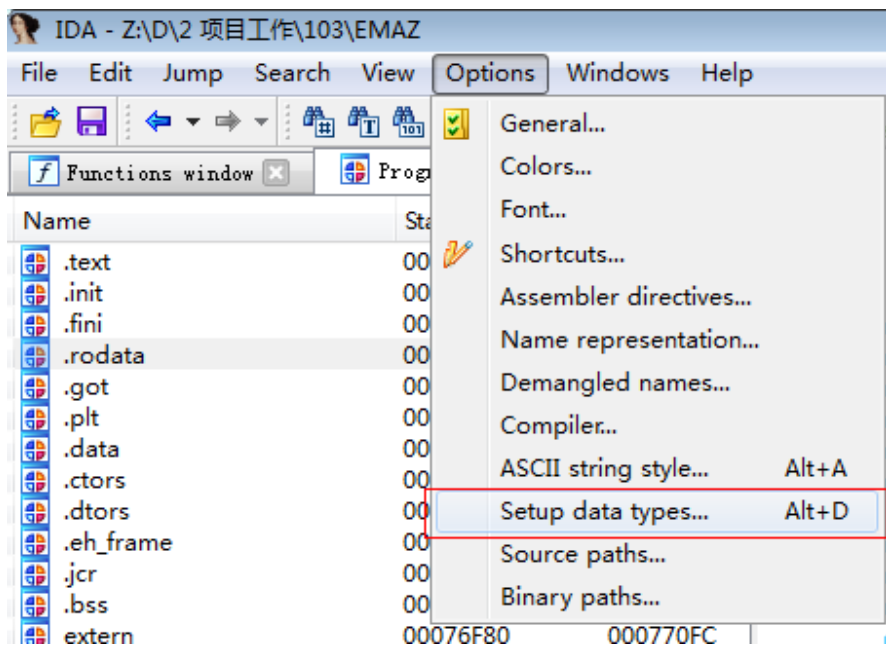


如果对数据文件进行了修改(错改)，但是不想保存，则勾选下方“**DON'T SAVE the database**”

三、IDA 工具的使用

• 数据格式转换

- IDA以什么方式来显示数据：整数、字符串、双精度数等。真实存在的数据并没有改变，只是换了一个样子展示自己。



建议每个文件首次打开时进行数据类型设置：勾选“float”、“double”

三、 IDA 工具的使用

- 常用快捷操作

- ① 搜索：**ALT+T**
- ② 数据格式切换：**D、Q、K、B、C、A**
- ③ 添加/编辑注释：**shift+:**
- ④ 上下层调用关系查看：**Xrefs graph to / from**
- ⑤ 窗口切换：**空格**
- ⑥ 查看上层调用：**X**
- ⑦ 简易计算器：**?**
- ⑧ 后退：**Esc**

注意，不是撤销，IDA中无撤销操作。

四、从汇编到C

• 认识代码段

```
.text:00013BA0      .global EMA2xAP_functions_execute_service
.text:00013BA0      EMA2xAP_functions_execute_service:      ? DATA XREF: .got:EMA2xAP_functions_execute_service_ptr↓o
.text:00013BA0      var_C          = -0xC
.text:00013BA0      save          %sp, -0x68, %sp
.text:00013BA4      sethi         %hi(loc_21800), %17
.text:00013BA8      call          sub_12888
.text:00013BAC      set           loc_21960, %17
.text:00013BB0      cmp           %i1, 0
.text:00013BB4      bne           %icc, loc_13BCC
.text:00013BB8      mov           %i1, %o5
.text:00013BBC      set           0x98, %g1
.text:00013BC4      ba           %xcc, loc_13C38
.text:00013BC8      ld            [%17+%g1], %o4      ? //%g=0x98, %17+%g1=355A0, %o4=235D8, _LLC24
                                   ? %o4 = "TTXAP_STATE_UNDEFINED;
.text:00013BCC      ? -----
.text:00013BCC      loc_13BCC:      ? CODE XREF: EMA2xAP_functions_execute_service+14↑j
.text:00013BCC      cmp           %i1, 1
.text:00013BD0      bne           %icc, loc_13BE8
.text:00013BD4      cmp           %i1, 2
.text:00013BD8      set           0x9C, %g1
.text:00013BE0      ba           %xcc, loc_13C38
.text:00013BE4      ld            [%17+%g1], %o4      ? //%g=0x9C, %17+%g1=355A4, %o4=235F0, _LLC25
                                   ? %o4 = "TTXAP_STATE_INPUT;
.text:00013BE8      ? -----
.text:00013BE8      loc_13BE8:      ? CODE XREF: EMA2xAP_functions_execute_service+30↑j
.text:00013BE8      bne           %icc, loc_13C00
.text:00013BEC      cmp           %i1, 3
.text:00013BF0      set           0xA0, %g1
```

函数名称

段标识

汇编指令

地址

标签

注释

四、从汇编到C

• 全局偏移表.got

记录全局变量（常量）的存储地址，%l7存放Got表的首地址

GLOBAL OFFSET TABLE

```
.text:00012888 sub_12888:                                ! CODE XREF: EMAZxAP_functions_get_config+8↓p
.text:00012888                                         ! EMAZxAP_functions_create_data_structure+8↓p ...
.text:00012888             retl
.text:0001288C             add     %o7, %l7, %l7
.text:0001288C ! End of function sub_12888
.text:0001288C
.text:00012890
.text:00012890 ! ===== S U B R O U T I N E =====
.text:00012890
.text:00012890 ! Attributes: bp-based frame
.text:00012890
.text:00012890             .global EMAZxAP_functions_get_config
.text:00012890 EMAZxAP_functions_get_config:           ! DATA XREF: .got:EMAZxAP_functions_get_config_ptr↓
.text:00012890             save     %sp, -0x60, %sp
.text:00012894             sethi    %hi(loc_22C00), %l7
.text:00012898             call    sub_12888             ! %o7=0x12898;
.text:0001289C             set     loc_22C70, %l7
.text:000128A0             set     0x190, %g1
.text:000128A8             ld      [%l7+%g1], %o0      ! //g=0x190, %l7+%g1=35698, %o0=233B8, __LLC0_0
.text:000128A8                                         ! %o0 = "EMAZ;
.text:000128AC             mov     2, %o1
.text:000128B0             set     0x17C, %g1
.text:000128B8             ld      [%l7+%g1], %o2      ! //g=0x17C, %l7+%g1=35684, %o2=23F10, __FUNCTION_
.text:000128B8                                         ! %o2 = "EMAZxAP_functions_get_config;
.text:000128BC             set     0x194, %g1
.text:000128C4             call    _THXAttrace
.text:000128C8             ld      [%l7+%g1], %o3      ! //g=0x194, %l7+%g1=3569C, %o3=233C0, __LLC1_0
.text:000128C8                                         ! %o3 = "> ();
.text:000128CC             cmp     %i1, 0
```

确定Got表首地址放入%l7

通过[%l7+Reg/Imm]访问全局变量/常量

四、从汇编到C

- **数据类型**

- 严格地来讲，IDA中是不存在数据类型的概念的。
- 要看数据的类型请从它的使用入手。
 - ldsb：很可能是一个字符(串)
 - ldd：很可能是一个double
 - 取地址操作时：（多级）指针
- 看IDA中的数据类型转换。

```
.rodata:00023258 _LLC2:      .byte 0x56 ? U           ? DATA XREF: .got:_LLC2_ptr↓o
.rodata:00023259      .byte 0x65 ? e
.rodata:0002325A      .byte 0x72 ? r
.rodata:0002325B      .byte 0x69 ? i
.rodata:0002325C aFicationFailed:.ascii "fication failed: '%s' not TRUE"<0>
```

四、从汇编到C

- 变量在IDA中的表达

外部变量在extern段中

```
extern:00077064                                     ? CODE XREF: _EMMIXAxFIL_lowpass_sy
extern:00077068                                     .extern ERXA_current_env
extern:00077068                                     ? DATA XREF: .got:ERXA_current_env_
extern:0007706C                                     .extern PLXAmem_free      ? CODE XREF: _PLXAmem_free↑j
extern:00077070                                     .extern TMXA_phase_req    ? CODE XREF: _TMXA_phase_req↑j
extern:00077074                                     .extern DDXA_var_string_alloc
```

全局变量(已定义初值) 在data段中

```
.data:000362A0 __gthread_active_0: word 0xFFFFFFFF
.data:000362A4      .byte 0
.data:000362A5      .byte 0
.data:000362A6      .byte 0
.data:000362A7      .byte 0
```

全局变量(未定义初值) 在bss段中

```
.bss:000365F8 unexpanded_file_name_14358: .space 1
.bss:000365F9      .space 1
.bss:000365FA      .space 1
.bss:000365FB      .space 1
.bss:000365FC      .space 1
.bss:000365FD      .space 1
.bss:000365FE      .space 1
.bss:000365FF      .space 1
```

四、从汇编到C

- 字符串常量在IDA中的表达

rodata段：Read Only Data 存储只读的（const）全局变量，通常字符串字面值就存在这个段

```
.rodata:00023240 ? =====
.rodata:00023240
.rodata:00023240 ? Segment type: Pure data
.rodata:00023240 .section .rodata? CONST
.rodata:00023240 .global _lib_version
.rodata:00023240 _lib_version: .byte 0 ! Alternative name is 'Drodata.rodata'
.rodata:00023241 .byte 0
.rodata:00023242 .byte 0
.rodata:00023243 .byte 1
.rodata:00023244 .byte 0
.rodata:00023245 .byte 0
.rodata:00023246 .byte 0
.rodata:00023247 .byte 0
.rodata:00023248 _LLC0: .ascii "EMAZ"<0> ! DATA XREF: .got:_LLC0_ptr↓o
.rodata:00023240 .align 0x10
.rodata:00023250 _LLC1: .ascii "> ()"<0> ! DATA XREF: .got:_LLC1_ptr↓o
.rodata:00023255 .align 4
.rodata:00023258 _LLC2: .word 0x56657269 ! DATA XREF: .got:_LLC2_ptr↓o
.rodata:0002325C aFicationFailed: .ascii "fication failed: '%s' not TRUE"<0>
.rodata:0002325C ! DATA XREF: frame_dummy+C↑o
.rodata:0002327B .align 0x10
.rodata:00023280 _LLC3: .ascii "cb_ptr != NULL"<0> ! DATA XREF: .got:_LLC3_ptr↓o
```


四、从汇编到C

• 变量在IDA中的表达 (2)

局部变量

在函数内部定义的非静态变量，作用域是函数内部；生命周期都是该函数的生命周期。在IDA中集中表示在函数名和汇编指令之间。命名方式一般为“var_ + 偏移地址”

```
.text:0001AC3C calc_new_expose_mirrormap:
```

```
.text:0001AC3C
```

```
.text:0001AC3C
```

```
.text:0001AC3C var_1984 = -0x1984
```

```
.text:0001AC3C var_1980 = -0x1980
```

```
.text:0001AC3C var_197C = -0x197C
```

```
.text:0001AC3C var_1978 = -0x1978
```

```
.text:0001AC3C var_1974 = -0x1974
```

```
.text:0001AC3C var_1970 = -0x1970
```

```
.text:0001AC3C var_196C = -0x196C
```

```
.text:0001AC3C var_1968 = -0x1968
```

```
.text:0001AC3C var_CB8 = -0xCB8
```

```
.text:0001AC3C var_20 = -0x20
```

```
.text:0001AC3C var_18 = -0x18
```

```
.text:0001AC3C var_10 = -0x10
```

```
.text:0001AC3C var_8 = -8
```

```
.text:0001AC3C
```

```
.text:0001AC3C save %sp, -0x1000, %sp
```

```
.text:0001AC40 inc -0x9E0, %sp
```

```
.text:0001AC44 sethi %hi(loc_1A800), %17
```

```
.text:0001AC48 call sub_18CB0
```

```
! CODE XREF:
```

```
! EMAZxAP_MC
```

变量和变量地址

假设一个变量是“int x = 0;”，则x在IDA中会以“var_xAddress”的方式出现。

“add %fp, var_xAddress, %o0”，则%o0中是x的地址——&x；

“ld [%fp+var_xAddress], %o0”，则%o0中存放的是x的值——x。

x本来是同一个变量，但是以不同的面目出现了两次，请不要遇到第一个面目的时候把x声明为“int *x = NULL;”，遇到第二个面目的时候声明成“int x = 0;”。

四、从汇编到C

- If语句

```
if(statement)
{
    .....
}
.....
```

```
text:0001482C
text:00014830
text:00014834
text:00014838
text:0001483C
text:00014840
text:00014844
text:00014848
text:0001484C
text:00014850
text:00014854
text:00014858
text:0001485C
text:00014860
text:00014864
text:00014868
text:0001486C
text:00014870
text:00014874
text:00014878
text:0001487C
```

loc_14874:

```
mov     %l0, %o0          ! stream
cmp     %o0, 0
be      loc_14874
sethi   %hi(0), %o1       ! mode
st      %o0, [%fp+linkError]
clr     [%fp+linkError+4]
bset    0x70, %o1         ! 32624
sethi   %hi(0), %o3
bset    0x74, %o3
sethi   %hi(0x53408000), %o0
bset    7, %o0
ld      [%l7+%o1], %o2    ! //"CNXAtaskInit failed
mov     0x172, %o4
ld      [%l7+%o3], %o3    ! %l7+%o3=32628, %o3=1F!
call    _ERXAllogError
add     %fp, linkError, %o1 ! mode
call    _exit
mov     1, %o0            ! stream

! CODE XREF: main+40fj
call    _ERXAsignalInstall
nop
```

四、从汇编到C

- If语句

```
if(statement)
{
    .....
}
else
{
    .....
}
.....
```

Assembly code for the if-else statement:

```
.text:00010678      sra      %g1, 24, %g1
.text:0001067C      cmp      %g1, 0x12
.text:00010680      bne      loc_106A0
.text:00010684      nop
.text:00010688      set      aBigEndian_, %00 ? "Big Endian.\n"
.text:00010690      call     _printf
.text:00010694      nop
.text:00010698      ba       loc_106B0
.text:0001069C      nop
.text:000106A0      ? -----
.text:000106A0      loc_106A0:
.text:000106A0      ? CODE XREF: main+20↑j
.text:000106A0      set      aLittleEndian_, %00 ? "Little Endian.\n"
.text:000106A8      call     _printf
.text:000106AC      nop
.text:000106B0      loc_106B0:
.text:000106B0      ? CODE XREF: main+38↑j
.text:000106B0      mov      0, %g1
.text:000106B4      mov      %g1, %i0
```

Annotations:

- A red box highlights the `cmp %g1, 0x12` instruction, with a red arrow pointing to the text `If %g1 == 0x12`.
- A red arrow points from the `bne loc_106A0` instruction to the `loc_106A0:` label.
- A red arrow points from the `ba loc_106B0` instruction to the `loc_106B0:` label.

四、从汇编到C

• switch case语句


.text:0001CEC0	set	0x184E8, %g2	
.text:0001CEC8	sub	%17, %g2, %g2	! %17=0x35508, %g2=%17-0x184E8=0x1D020;
.text:0001CECC	sll	%i0, 2, %i0	! switch i0
.text:0001CED0	ld	[%g2+%i0], %g3	! %g3 = [%g2+4*i0] = [0x1D020+4*i0]
.text:0001CED4	jmp	%g2+%g3	! ba 0x1D020+[0x1D020+4*i0]
.text:0001CED4			! case i0 = 0: ba 0x1CEDC;
.text:0001CED4			! case i0 = 1: ba 0x1CEE8;
.text:0001CED4			! ...
.text:0001CED8			
.text:0001CEDC	nop		
.text:0001CEDC	mov	0x1A, %g2	
.text:0001CEE0	ba	%xcc, loc_1D010	
.text:0001CEE4	mov	0, %g3	
.text:0001CEE8			
.text:0001CEE8			
.text:0001CEEC			
.text:0001CEF0			
.text:0001CEF4			
.text:0001CEF4			
.text:0001CEF8			
.text:0001CEFC			
.text:0001CF00			
.text:0001CF00	mov	2, %g2	
.text:0001CF04	ba	%xcc, loc_1D010	
.text:0001CF08	mov	3, %g3	
.text:0001CF0C			
.text:0001CF0C	mov	3, %g2	
.text:0001CF10	ba	%xcc, loc_1D010	
.text:0001CF14	mov	4, %g3	
.text:0001CF18			
.text:0001CF18	mov	4, %g2	
.text:0001CF1C	ba	%xcc, loc_1D010	
.text:0001CF20	mov	5, %g3	
.text:0001CF24			

.text:0001D010	.word	0xFFFFFEB8, 0xFFFFFEC8, 0xFFFFFED4, 0xFFFFFEE0, 0xFFFFFEEC
.text:0001D020	.word	0xFFFFFEF8, 0xFFFFFF04, 0xFFFFFF10, 0xFFFFFF1C, 0xFFFFFF28
.text:0001D020	.word	0xFFFFFF34, 0xFFFFFF40, 0xFFFFFF4C, 0xFFFFFF58, 0xFFFFFF64
.text:0001D020	.word	0xFFFFFF70, 0xFFFFFF7C, 0xFFFFFF88, 0xFFFFFF94, 0xFFFFFFA0
.text:0001D020	.word	0xFFFFFFAC, 0xFFFFFFB8, 0xFFFFFFC4, 0xFFFFFFD0, 0xFFFFFFDC
.text:0001D088	.word	0xFFFFFEE8

四、从汇编到C

• 循环语句

```
for(...)
{
...
}
.....
```



```
.text:000178CC      add     %fp, var_80, %g2
.text:000178D0      mov     -1, %g3
.text:000178D4      st      %g3, [%fp+var_9C]
.text:000178D8      mov     -1, %l6
.text:000178DC      mov     0, %g1
.text:000178E0      loc_178E0:
.text:000178E0      ld      [%g2], %g4
.text:000178E4      cmp     %g4, 1
.text:000178E8      hne,a   %icc, loc_1791C
.text:000178EC      inc     %g1
.text:000178F0      inc     %l6
.text:000178F4      cmp     %g1, 4
.text:000178F8      mov     0, %g4
.text:000178FC      movle   %icc, 1, %g4
.text:00017900      xnor    %g0, %g3, %o5
.text:00017904      cmp     %g0, %o5
.text:00017908      subc    %g0, -1, %o5
.text:0001790C      bset    %o5, %g4
.text:00017910      loc_17910:
.text:00017910      btst    0xFF, %g4
.text:00017914      movne   %icc, %g1, %g3
.text:00017918      inc     %g1
.text:0001791C      loc_1791C:
.text:0001791C      cmp     %g1, 9
.text:00017920      bne     %icc, loc_178E0
.text:00017924      inc     4, %g2
.text:00017928      st      %g3, [%fp+var_9C]
```

! CODE XREF: calc_results+6FC↓j

! DATA XREF: __ctzsi2+10↓o

! CODE XREF: calc_results+6C4↑j

四、从汇编到C

- 函数的参数

函数的参数依次存放在%i0, %i1, %i2, %i3, %i4, %i5, %fp+0x5C, %fp+0x60, %fp+0x6C ...

```
emgmap_get_file_name:                                ! CODE XREF: EMGMAP_check_testlog_name+3FC↓p
                                                       ! EMGMAP_retrieve_testlog+1A8↓p ...

var_14                = -0x14
var_8                 = -8
var_4                 = -4

save                %sp, -0x70, %sp
sethi               %hi(loc_48400), %17
call                sub_435C8
set                 loc_48474, %17
set                 0x2E746C67, %g1
st                  %g1, [%fp+var_8]
clrb                [%fp+var_4]
add                 %fp, var_8, %g1
st                  %g1, [%sp+0x5C]
mov                 %i1, %00
set                 0x11B0, %g1
ld                  [%17+%g1], %01    ! //%g=0x11B0, %17+%g1=8EDB0, %01=73908, _LLC137_1
                                   ! %01 = "service_data/%s/%s%s_%s";

mov                 %i0, %02
mov                 %i0, %03
add                 %i0, 8, %04
call                _sprintf
add                 %i0, 0x10, %05
```

四、从汇编到C

- 函数的调用

call指令表示函数调用，参数依次存入到%o0, %o1, %o2, %o3, %o4, %o5 , %sp+0x5C, %sp+0x60, %sp+0x6C ...

```
loc_45754:                                † CODE XREF: EMGMAP_test_name_cmd+BC8†j
    ld      [%i1], %g1
    smul    %g1, 0x718, %g1
    add     %i0, %g1, %i0
    call    _EMGMAP_display_result_table
    ld      [%i0+0x720], %o0
    cmp     %o0, 0
    bne, pn %icc, loc_45660
    st      %o0, [%fp+var_4]
    set     0x1000, %g1
    ba      %xcc, loc_456B8
    ld      [%i7+%g1], %o4 † //g=0x1000, %i7+%g1=8ECD0, %o4=73268, _LLC72_3
```

函数调用的返回值放在%o0中。

四、从汇编到C

- 函数的返回

return语句表示函数返回、如有返回值则放在%i0(%o0)。

```
loc_2EC40:                                ! CODE XREF: EMGMFQ_start_test+50↑j
      set     0x564, %g1
      ld      [%i7+%g1], %o0             ! //%g=0x564, %i7+%g1=8E164, %o0=6E040, _LLC0_5
                                           ! %o0 = "EMGM";
      mov     1, %o1
      set     0x5B0, %g1
      ld      [%i7+%g1], %o2             ! //%g=0x5B0, %i7+%g1=8E1B0, %o2=6E1C0, __FUNCTION__21428
                                           ! %o2 = "EMGMFQ_start_test";
      set     0x56C, %g1
      ld      [%i7+%g1], %o3             ! //%g=0x56C, %i7+%g1=8E16C, %o3=6E050, _LLC2_5
                                           ! %o3 = "< () = %R";
      call    _THXAtrace
      mov     %i0, %o4
      return  %i7+8
      nop
```

%i7 = 函数被调用处的地址 , return %i7+8 等价于 :

```
    jmp  %i7+8,%g0
    restore
```


四、从汇编到C

- 变量名、参数名如何确定？

IDA中没有变量名的信息，它操作的无非是寄存器、地址或者立即数，只能从上下文语中去推断。

```
set     0x464, %g1
ld      [%17+%g1], %o2    ! //%g=0x464, %17+%g1=3596C, %o2=24C18, _LLC2_4
                        ! %o2 = "EMAZxAP_measure_cycle;
set     0x468, %g1
ld      [%17+%g1], %o3    ! //%g=0x468, %17+%g1=35970, %o3=24C30, _LLC3_4
                        ! %o3 = "> (cycle_nr=%d);
call    THXAtrace
ld      [%fp+arg_44], %o4
```

fp+arg_44 : cycle_nr

当名称暂时不能确定时，按类型前缀+偏移地址进行命名：

```
int      g_i0x3FF30;
double   g_d0x3FF30;
int      *g_pi0x400A8;
double   *g_pd0x400A8;
```

```
typedef struct
{
    int cycle_nr;           //0x0
    int chuck;             //0x4
    double residuals_without_fit; //0x8
    double fit_residuals;   //0x10
    double spec_sigma_fit;  //0x18
    char s0x20[0x100];      //0x20
} EMAZ_fit_residuals; //size = 0x120
```

```

text:0004AE34 .global double_list_create
text:0004AE34 double_list_create: ? CODE XREF: _double_list_create↓j
text:0004AE34 save    %sp, -0x60, %sp
text:0004AE38 sethi   %hi(loc_42C00), %l7
text:0004AE3C call    sub_4AD8C
text:0004AE40 set     loc_42DC4, %l7
text:0004AE44 mov     %i0, %o0
text:0004AE48 call    calloc
text:0004AE4C mov     8, %o1
text:0004AE50 cmp     %o0, 0
text:0004AE54 bne     %icc, loc_4AEB4
text:0004AE58 mov     %o0, %l0
text:0004AE5C set     0x1684, %g1
text:0004AE64 ld      [%l7+%g1], %o0 ? //%g=0x1684, %l7+%g1=8F284, %o0=74FD8, _LLC0_13
text:0004AE64 ? %o0 = "%s: system error:\n\tFailed to allocate %u bytes of memory ;
text:0004AE68 set     0x1690, %g1
text:0004AE70 ld      [%l7+%g1], %o1 ? //%g=0x1690, %l7+%g1=8F290, %o1=75048, _LLC3_13
text:0004AE70 ? %o1 = "double_list_create;
text:0004AE74 call    _ERXAmakeContext
text:0004AE78 sll     %i0, 3, %o2
text:0004AE7C mov     %o0, %o5
text:0004AE80 set     0x454D6401, %o0
text:0004AE88 mov     0, %o1
text:0004AE8C set     0x168C, %g1
text:0004AE94 ld      [%l7+%g1], %o2 ? //%g=0x168C, %l7+%g1=8F28C, %o2=75028, _LLC2_13
text:0004AE94 ? %o2 = "../inc/bld/EMGMDM_object.c;
text:0004AE98 mov     0xA0, %o3
text:0004AE9C call    _ERXAllogExceptionSingleLink
text:0004AEA0 mov     0, %o4
text:0004AEA4 call    _free
text:0004AEA8 mov     %l0, %o0
text:0004EAC return   %i7+8
text:0004AEB0 hset    1, %o0
text:0004AEB4 ? -----
text:0004AEB4 loc_4AEB4: ? CODE XREF: double_list_create+20↑j
text:0004AEB4 st      %o0, [%i1]
text:0004AEB4 return   %i7+8
text:0004AEB8 mov     0, %o0
text:0004AEB8 ? End of function double_list_create

```

开辟函数栈帧

确定Got表首地址放入%l7

calloc(%i0,8); %i0为unsigned int

判断calloc函数返回

calloc返回为0分支处理

calloc返回不为0分支处理
将返回值赋值到[%i1]

四、从汇编到C

- 解析后的C代码

```
int double_list_create(unsinged int nr, double **pptr)
{
    int ret = 0;
    char *error_text = NULL;
    double *ptr = NULL;

    ptr = (double *)calloc(nr, sizeof(double));
    if (NULL == ptr)
    {
        error_code = 0x454D6401;
        error_text = ERXAmakeContext("%s: system error:\n\tFailed to allocate %u bytes of memory",
            "double_list_create", nr);
        ERXalogExceptionSingleLink(error_code, 0, "../inc/bld/EMGMDM_object.c", 0xA0, 0, error_text);
        free(ptr);
        ret = 1;
    }
    else
    {
        *pptr = ptr;
    }

    return ret;
}
```

• 练习

解析如下函数：

```
.text:00010688
.text:00010688 main:
.text:00010688
.text:00010688 var_3C
.text:00010688 var_38
.text:00010688 max
.text:00010688 min
.text:00010688 number
.text:00010688
.text:00010688
.text:0001068C
.text:00010690
.text:00010694
.text:0001069C
.text:000106A0
.text:000106A4
.text:000106A8
.text:000106B0
.text:000106B4
.text:000106B8
.text:000106BC
.text:000106C4
.text:000106C8
.text:000106CC
.text:000106D0
.text:000106D8
.text:000106E0
.text:000106E4
.text:000106EC
.text:000106F0
.text:000106F4
.text:000106F8
.text:000106FC
.text:00010700
.text:00010704
```

```
.rodata:000107A0 aMax:
.rodata:000107A4
.rodata:000107A5
.rodata:000107A6
.rodata:000107A7
.rodata:000107A8 aSDSLFSLF:
.rodata:000107BE
.rodata:000107C0 aNumber:
.rodata:000107C7
.rodata:000107C8 aMin:
.rodata:000107CC
.rodata:000107D0 dbl_107D0:
.rodata:000107D8 dbl_107D8:
.rodata:000107D8
= -0x14

save %sp, -0x98, %sp
mov 0x64, %g1
st %g1, [%fp+number] ! number = 0x64;
set dbl_107D0, %g1 ! [%g1] = -1000.0;
ld [%g1], %f8
ld [%g1+4], %f9
std %f8, [%fp+min] ! min = -1000.0;
set dbl_107D8, %g1 ! [%g1] = 1000.0;
ld [%g1], %f8
ld [%g1+4], %f9
std %f8, [%fp+max] ! max = 1000.0;
set aMax, %g1 ! %g1 = "max";
st %g1, [%sp+0x98+var_3C] ! 按Q可转换成[sp+0x5C]
ldd [%fp+max], %f8
std %f8, [%sp+0x60]
set aSDSLFSLF, %o0 ! "%s=%d, %s=%lf, %s=lf\n"
set aNumber, %o1 ! "number"
ld [%fp+number], %o2
set aMin, %o3 ! "min"
ldd [%fp+min], %o4
call _printf
nop
mov 0, %g1
mov %g1, %i0
ret
restore
```

- 练习

```
#include <stdio.h>

int main()
{
    int number = 100;
    double min = -1000.0;
    double max = 1000.0;

    printf("%s=%d, %s=%lf, %s=%lf\n", "number", number, "min", min, "max", max);

    return 0;
}
```

五、例子

- 待解析文件



- 解析的结果



学习资料

- SPARCOverview.pdf
- SPARC V8.pdf
- SPARC V9.pdf
- Sparc指令集.xls
- Understanding stacks and registers in the Sparc architecture(s).doc
- IDA Pro权威指南第2版.pdf

Thank you !