四、程序的逆向识别

程序的逆向识别

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int a, b = 5;
6      scanf("%d", &a);
7
8      if(a == 0)
9          a = 8;
10
11     return a + b;
12 }
```

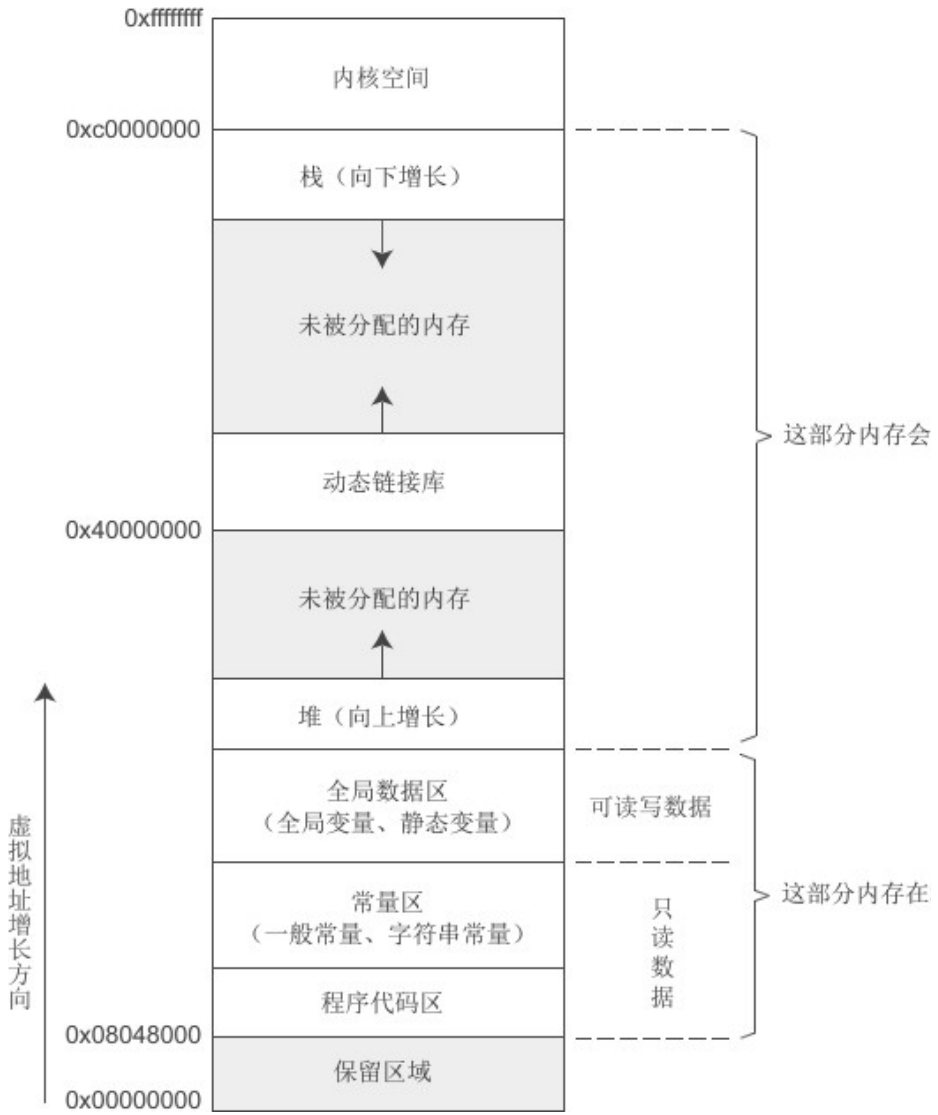
```
; int __cdecl main(int argc, const char **argv, const char **envp)
                public _main
_main           proc near               ; CODE XREF: ___mingw_CRTStartup

argc            = dword ptr  8
argv            = dword ptr  0Ch
envp            = dword ptr  10h

                push    ebp
                mov     ebp, esp
                and     esp, 0FFFFFFF0h ; Logical AND
                sub     esp, 20h        ; Integer Subtraction
                call    ___main         ; Call Procedure
                mov     dword ptr [esp+1Ch], 5
                lea     eax, [esp+18h]  ; Load Effective Address
                mov     [esp+4], eax
                mov     dword ptr [esp], offset aD ; "%d"
                call    _scanf          ; Call Procedure
                mov     eax, [esp+18h]
                test    eax, eax        ; Logical Compare
                jnz     short loc_40137A ; Jump if Not Zero (ZF=0)
                mov     dword ptr [esp+18h], 8

loc_40137A:                             ; CODE XREF: _main+30↑j
                mov     edx, [esp+18h]
                mov     eax, [esp+1Ch]
                add     eax, edx        ; Add
                leave                   ; High Level Procedure Exit
                retn                    ; Return Near from Procedure
_main           endp
```

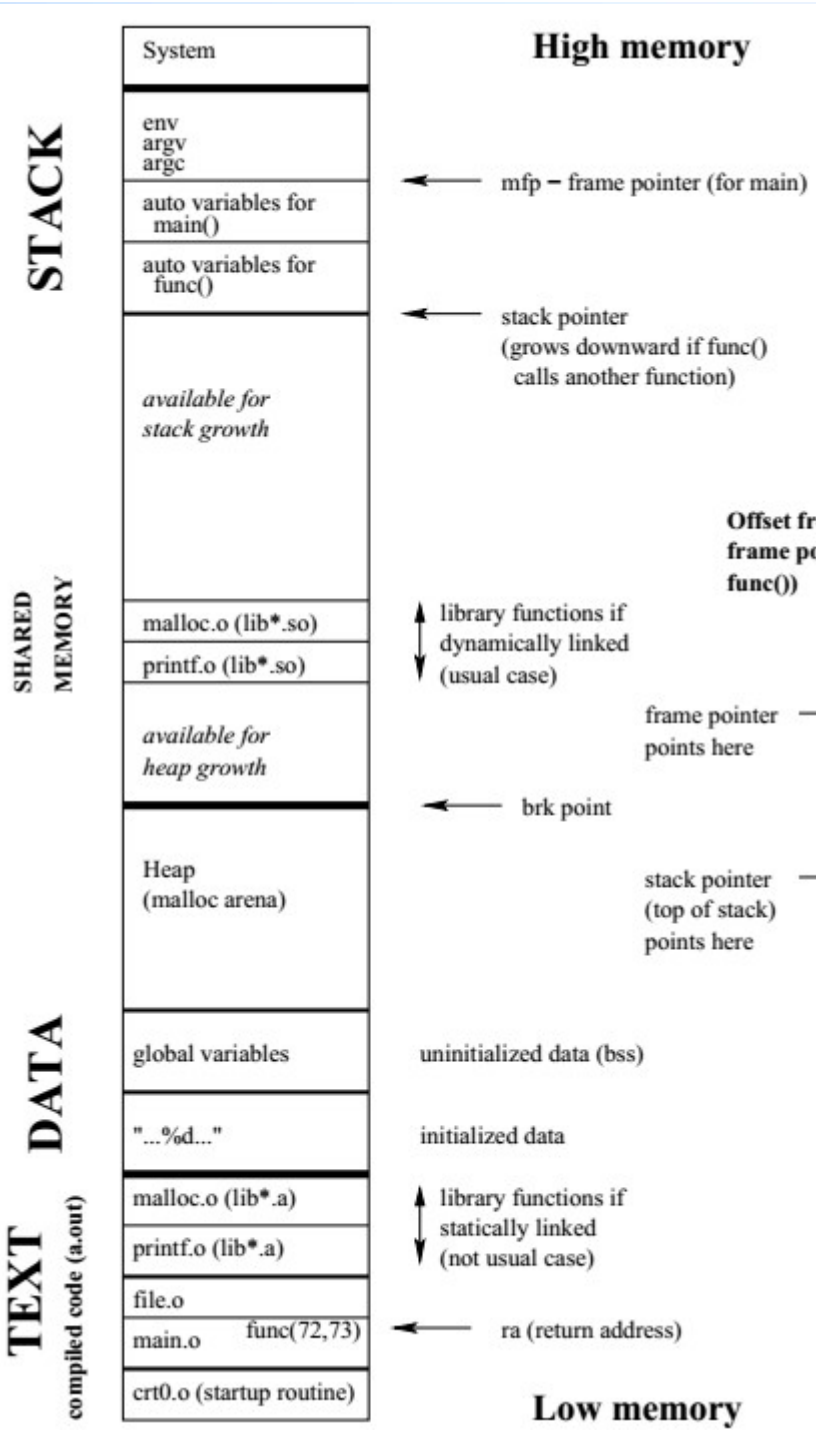
如何从汇编中识别-栈帧



- 栈区(Stack)：由编译器自动分配释放，其操作方式类似于数据结构中的栈，用于存放函数的形参、返回地址、返回数据，局部变量的值等
- 堆区(Heap)：一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收，其存储方式类似于链表（malloc、calloc、realloc、free）
- 全局（静态区）(Static)：程序结束后由系统释放，用于存放全局变量、静态变量，已初始化的全局变量和未初始化的静态变量放在一块区域，未初始化的全局变量和未初始化的静态变量放在相邻另一块区域
- 文字常量区：程序结束后由系统释放，用于存放常量字符串等
- 程序代码区：存放函数体（类成员函数和全局函数）的二进制代码

如何从汇编中识别-栈帧

```
1  void func(int x, int y)
2  {
3     int a;
4     int b[3];
5     /* no other auto variable */
6    ...
7  }
8  void main()
9  {
10     ...
11     func(72,73);
12     ...
13  }
```



如何从汇编中识别-栈帧

栈帧（stack frame)

- 栈帧也常被称为“**活动记录**”(activation record)，是编译器用来实现函数调用的一种数据结构

- 从逻辑上讲，栈帧就是一个**函数执行的环境**，包含所有与函数调用相关的数据：主要包括函数参数、函数中的局部变量、函数执行完后的返回地址，被函数修改的需要恢复的任何寄存器的副本

- **栈是从高地址向低地址延伸的**。每个函数的每次调用，都有它自己独立的一个栈帧，这个栈帧中维持着所需要的各种信息

如何从汇编中识别-栈帧

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

栈帧（stack frame)

- 寄存器ebp用来指向当前的栈帧的底部（高地址）
- 寄存器esp用来指向当前的栈帧的顶部（低地址）
- 即在函数调用执行过程中，寻找所需参数或变量信息时使用寄存器ebp来寻址，因为寄存器esp的值是经常变化的，而寄存器ebp的值对一个函数的栈帧来讲是不变的

如何从汇编中识别-栈帧

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- caller调用者
- callee被调用者，即子函数
- arguments：子函数参数，调用前入栈
- return address： 子函数调用前将待执行下一条指令地址保存在返回地址中，待函数调用结束后，返回到调用者继续执行
- eip 指令指针指向下一条指令地址
- ebp 基址指针指向栈帧底（高地址）
- esp 栈指针指向栈顶（低地址）

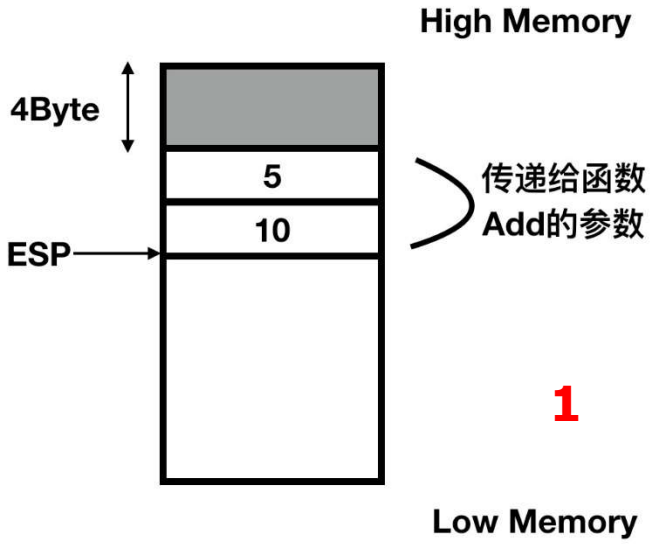
如何从汇编中识别-栈帧

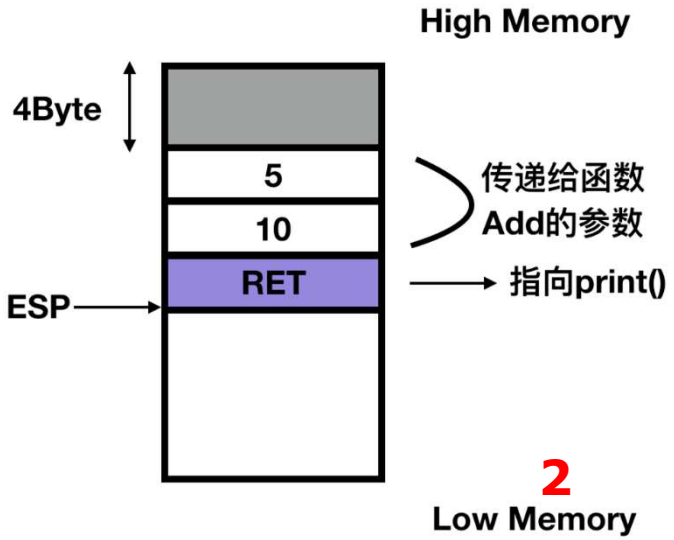
哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- caller（调用者）规则：
  - 子函数参数入栈，从右向左
  - call指令，将下一条指令地址入栈（push eip），并无条件跳转
  - 子函数返回，返回值在eax中
- callee（被调用者）规则：
  - 保存caller的栈基址，设定callee新的栈基址为当前栈指针（push ebp; mov ebp, esp）
  - 为局部变量分配栈空间（sub esp, 123）
  - 执行函数，结果保存在eax中
  - 执行leave复合指令，清除当前栈帧，恢复到调用者栈帧（mov esp，ebp; pop ebp）
  - 执行ret指令（pop eip）

如何从汇编中识别-栈帧

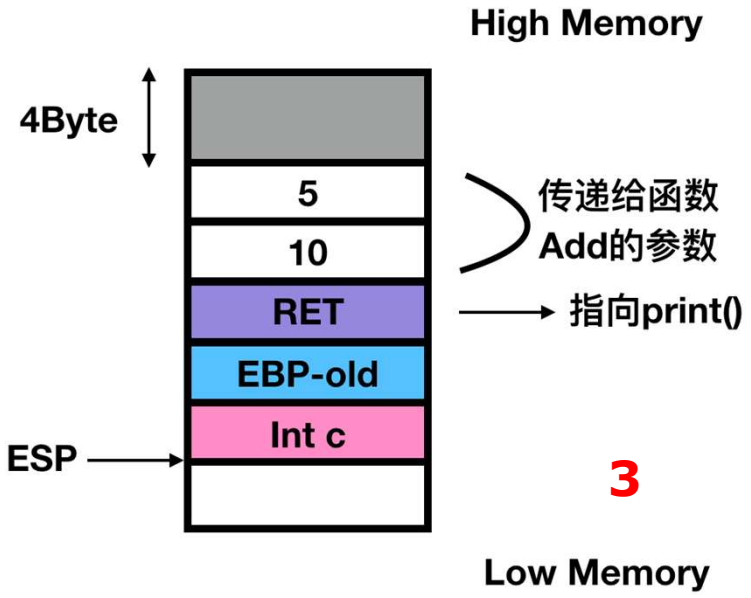

```
1  int add(int a, int b)
2  {
3      int c;
4      c = a + b;
5      return c;
6  }
7
8  void main()
9  {
10     add(10, 5);
11     printf("hello world!");
12 }
```





如何从汇编中识别-栈帧


```
1  int add(int a, int b)
2  {
3      int c;
4      c = a + b;
5      return c;
6  }
7
8  void main()
9  {
10     add(10, 5);
11     printf("hello world!");
12 }
```



案例分析：观察栈帧的动态变化


```
1   #include <stdio.h>
2
3   long add(long a, long b)
4   {
5       long x = a, y = b;
6
7       return (x+y);
8   }
9
10  int main(int argc, char* argv[])
11  {
12      long a = 1, b = 2;
13      printf("%d\n", add(a, b));
14
15      return 0;
16  }
```

案例分析：缓存区溢出

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
 1  # include <stdio.h>
 2
 3  int read_req(void)
 4  {
 5      char buf[128];
 6      int i;
 7      gets(buf);
 8      i = atoi(buf);
 9      return i;
10  }
11
12  int main(int ac, char **av)
13  {
14      int x = read_req();
15      printf("x=%d\n", x);
16  }
```