**1. ZF:Zero Flag 零标志 SF=Signed Flag 负数标志 CF：Carry Flag 进位标志 OF:Overflow Flag 溢出标志 【CF无符号数溢出，OF有符号数溢出】**

**注意：在加法中CF是进位，而在减法中，CF是借位 00000110 – 11111111 需要借位**

**所以答案都是 ZF=0 SF=0 CF=1 OF=0**

**判断CF的本质方法： (unsigned) t < (unsigned) u ，如果成立，则CF=1。**

**2. D ： 算术，逻辑，TEST/CMP这三类操作会影响条件码**

**3. D ： leaq不会改变条件码，mov指令一般也不会**

**4. C ： mov访问内存，leaq加载内容**

**5. A： JMP跳转，CALL调用，RET返回**

**6. A：push压栈，%rsp减少；call调用，%rsp压入返回地址；**

**7. D：禁止内存到内存**

**8.** **D**

**9.** **C（test相当于and） test和cmp指令只改变标志位，不会改变值**

**10.** **C ：加载内存地址常用leaq指令，取值采用mov**

**11.** **A 没有出现在分支中的标号（4和6跳转到默认分支）对应%rdi 3 和 6**

**12.** **D ： 1 char +3 对齐+4 int+1 char+ 3 尾部对齐 = 12**

**13.** **D ： %rdi %rsi %rdx %rcx %r8 %r9 ,后面参数才存到栈帧**

**14.** **A “leal 7(%eax, %eax, 4), %ebx”**

**15.** **C %eax=x+y ,%edx= y-x ，当x>=y时，把%edx传送给%eax，否则返回%eax**

**16.** **A**

**17.** **A 先分配栈帧，再压栈**

**18.** **D**

**19.** **B**

**20. 为什么把%rdx存到%r12: Q可能修改 %rdx 的值**

**为什么最后是16(%r12)：没看懂，感觉是12**

表格

AI 生成的内容可能不正确。

**21. 易错： mov指令不会改变条件码，只要中间没有算术/逻辑/CMP/TEST,就不会改变判断结果，所以第一次判断还是%rdi，第二次判断是f(n)**

表格

AI 生成的内容可能不正确。

**return a**

**22.**

图形用户界面, 表格

AI 生成的内容可能不正确。

**23.**

文本, 信件

AI 生成的内容可能不正确。

**24.**

图片包含 表格

AI 生成的内容可能不正确。

**25.** 函数传递引用类型本质上传的是指针，所以需要使用(%rdi)来访问

图形用户界面

AI 生成的内容可能不正确。

**26.**

图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

**27.**

表格

AI 生成的内容可能不正确。

**28.**

图示

AI 生成的内容可能不正确。

**29. 注意类型 long，不要先入为主写成int**

图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

**30.**

表格

AI 生成的内容可能不正确。

**31.** **对齐方式：0x108 0x118 0x128**

**32.** **C：union的特性：共用一块内存，同一时间只能存储并访问其中的一个成员，同时unsigned和float能同时表示的也只有0**

**33. 24**

**34.**

文本

AI 生成的内容可能不正确。

**35.** 参考答案：

int loop (int a[ ], int n) {

int i, sum;

sum = 0;

for ( i = 0 ; i < n ; i++ ) {

sum += (a[i] < 0 ? a[i] + 3 : a[i]) >> 2;

或

(a[i] >=0 ? a[i]: a[i] + 3) /4;

}

return sum;

}

参考答案：

func:

subq $0x10, %rsp # 分配栈空间

movl $0x09, 12(%rsp) # arr[3] = 9

movl $0x1, 8(%rsp)

movl $0, 4(%rsp) # arr[1] = 0

movl $0x2, (%rsp) # arr[0] = 2

movl $0, %eax

movl $0 %ecx

.L1

addl (%rsp, %eax, 4), %ecx

addl $1, %eax

cmpl $4, %eax

jl .L1

addl %ecx, (%rdi) # 将累加和存放于地址sum处的内存空间

addq $0x10, %rsp

ret

**36.**

**第一行变量a的作用域是全局，第三行变量a的作用域是函数proc内。**

**局部静态变量：**只会初始化一次（默认值为 0）；在后续的函数调用中保持上次的值；

作用域仍是函数内部，但生命周期是整个程序运行期间

**第一次：12**

**第二次：17**

**37.** **7 3 2**

**38.**

表格

AI 生成的内容可能不正确。

**39. H = 15 J = 10**

**40.** 参考答案：‌

1、前6个整型参数依次通过：%rdi（a）、%rsi（b）、%rdx（c）、%rcx（d）、%r8（e）、%r9（f）。

2、g位于16(%rbp)，h位于24(%rbp)。因为调用时返回地址（8字节）和保存的%rbp（8字节）已压栈，故第7个参数g在%rbp+16，第8个参数h在%rbp+24。

subq $32, %rsp为8个参数副本（每个int占4字节）预留空间，共32字节

3、main调用func时需将第7、8个参数（7和8）压栈（注意x86-64栈向下增长，参数按从右到左压栈）。

栈帧结构（从高地址到低地址）：

| ... |

| h=8 | ← %rbp+24

| g=7 | ← %rbp+16

| 返回地址 | ← %rbp+8

| 保存的%rbp | ← %rbp

| local1~local4 | ← %rbp-4 ~ %rbp-32

**41.**