姓名： 学号： 成绩：

1. 现有代码： int i=0xab cd ef 01;

short si=i;

请问代码执行后，变量si的数值表示为（十六进制）？

答：直接截断取低16位，因此si的数值表示为0xef 01。

2. 如果int i=0x86 23 11 32，&i=0x400320，请问地址0x400322 地址上的那个字节存储的数值是？

答：

* 大端：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址 | 0x400320 | 0x400321 | 0x400322 | 0x400323 |
| 值 | 0x86 | 0x23 | 0x11 | 0x32 |

地址0x400322所对应内存上的那个字节存储的数值是为0x11。

* 小端：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址 | 0x400320 | 0x400321 | 0x400322 | 0x400323 |
| 值 | 0x32 | 0x11 | 0x23 | 0x86 |

地址0x400322所对应内存上的那个字节存储的数值是为0x23。

3. 考虑一种遵从IEEE规范的新浮点格式，包含3个阶码位和3个小数位（即该浮点数不考虑符号位，只用来表示正数）。请回答下列问题。

1）除0和Infinity外，该浮点数能表示的数值范围为多少？

‌最小正数‌：非规格化数（denormal number）的最小非零值，对应阶码E=0（二进制000），小数位f=001（二进制），尾数M=0.001₂ = 1/8，实际指数exp = emin = 1 - bias，其中bias = 2^{3-1} - 1 = 3，因此exp = 1 - 3 = -2。数值 = M × 2^{exp} = (1/8) × 2^{-2} = (1/8) × (1/4) = 1/32。

非规格化数的最大，(7/8) × (1/4) = 7/32。

‌最大有限正数‌：规格化数（normal number）的最大值，对应阶码E=6（二进制110），小数位f=111（二进制），尾数M=1 + f = 1 + 7/8 = 15/8，实际指数exp = E - bias = 6 - 3 = 3。数值 = M × 2^{exp} = (15/8) × 2^3 = (15/8) × 8 = 15。规格化数的最小值与最大值分别是 0.25和15。

故该浮点数能表示的数值范围为1/32 15

2）尝试填下以下表格的空白处。如果一个数值太大而无法表达，使用infinity的表达式；如果一个数值太小而无法表达，使用0的表达式。

|  |  |
| --- | --- |
| 二进制表达 | 十进制数值 |
| 011000 | 1 |
| 111000 | 17 |
| 110001 | 9 |
| 110010 | 9+1/2 |

4.（结构体+函数+控制）已知node 结构体定义如下

struct node{

long a;

struct node \*next;

}

请对以下init函数进行逆向分析，写出其C代码

Init:

movl $12,$eax

jmp .TestExprStat

.Loop:

addq (%rdi),%rax

movq 8(%rdi),%rdi

.TestExprStat:

testq %rdi,%rdi

jne .Loop

ret

答：

long Init(struct node \*p) {  
 long res = 12;  
 while(p != NULL) {  
 res += p->a;  
 p = p->next;  
 }  
 return res;  
}

5. 考虑以下C函数调用和对应的x86-64汇编代码片段：

void func(int a, int b, int c, int d, int e, int f, int g, int h) {

// ... 其他操作

}

int main() {

func(1, 2, 3, 4, 5, 6, 7, 8);

return 0;

}

对应的x86-64汇编（部分）如下：

func:

pushq %rbp

movq %rsp, %rbp

subq $32, %rsp

movl %edi, -4(%rbp) ; a

movl %esi, -8(%rbp) ; b

movl %edx, -12(%rbp) ; c

movl %ecx, -16(%rbp) ; d

movl %r8d, -20(%rbp) ; e

movl %r9d, -24(%rbp) ; f

movl 16(%rbp), %eax ; g

movl %eax, -28(%rbp)

movl 24(%rbp), %eax ; h

movl %eax, -32(%rbp)

; ... 计算局部变量

‌问题：‌

1）在x86-64调用约定中，前6个整型参数通过哪些寄存器传递？请按顺序列出。

2）‌栈帧布局‌：

* 在func的栈帧中，参数g和h的存储位置相对于%rbp的偏移量是多少？为什么？
* 为什么subq $32, %rsp为局部变量预留了32字节空间？

‌3）调用前后栈指针变化‌：

* 在main调用func之前，栈指针%rsp需要如何调整？
* 画出调用func时的完整栈帧结构（包括返回地址、保存的%rbp、参数和局部变量）。

‌

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4个局部变量（每个int占4字节）预留空间，共16字节，剩余16字节可能用于对齐或临时存储。

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

6. 对于以下myfile.c程序代码

int hello1;

char myfun(int a)

{ int b=10;

hello1=b+a;

hello1=hello1\*f1(a);

printf(“%d\n”,hello1);

return hello1;

}

1）请指出那些符号是链接器符号，哪些符号需要重定位？

2）如果hello1改为static int hello1，其符号属性会发生什么变化？它会被分配到哪个ELF节？

3）ELF是什么？请画出myfile.o的ELF文件布局（包括节头表、.text、.data、.bss、.symtab、

.rel.text等），并标注hello1和myfun的存储位置。

4）解释f1是如何重定位的？

参考答案：

1）链接器符号: hello1，myfun，f1，printf；需要重定位: hello1，f1，printf

2）改为static int hello1后，hello1变为局部符号‌（仅当前文件可见），不再出现在全局符号表中，仍存储在.bss节（未初始化全局/静态变量）。

3）ELF是描述目标文件/可执行文件的标准格式，包含代码、数据、符号表等结构化信息，‌关键组成‌有：ELF头：标识文件类型（ET\_REL表示可重定位目标文件），节头表：描述各节（如.text、.bss）的偏移量、大小等属性；程序头表：可执行文件的段加载信息（目标文件无此表）。

4）‌编译阶段‌：

编译器发现f1未定义，生成一条调用指令（如call 0x0），地址暂填0，并在.rel.text中记录重定位条目：

* 类型：R\_X86\_64\_PC32（相对地址调用）。
* 符号：f1。
* 偏移：调用指令在.text中的位置。

‌链接阶段‌：

* 链接器扫描其他模块（如lib.a或共享库），找到f1的定义地址。
* 根据.rel.text条目，修改.text中call 0x0的偏移量，计算f1的实际相对地址：

公式：目标地址 - 下条指令地址。

7. 对于以下代码：

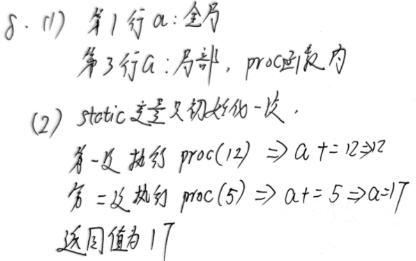
1. int a=100;
2. short proc(int inputarg)
3. { int a;

a+=inputarg;

return a+0;

1. }

请说明边第1行变量a和3行定义变量a的作用域。如果将第3行变量类型修改为static int，则第一次调用函数proc时传入参数inputarg的值为12，此时返回值为多少？再次调用时传入inputarg为5，则第二次调用的返回值是多少？



8. 已知一个硬盘容量为1TB，由两个盘（4个盘面）构成，每个盘面由3200个磁道构成，每个磁道平均有128个扇区。由于磁记录密度的提升，新工艺维持相同的容量下，只需要一个盘（2个盘面），每个盘面的磁道数为3200，则新工艺要求每个磁道多少个扇区？

参考答案：

‌旧工艺参数‌：

盘面数 = 4

磁道数/盘面 = 3200

扇区数/磁道 = 128

总容量 = 4×3200×128×S（S为扇区大小）

‌新工艺参数‌：

盘面数 = 2

磁道数/盘面 = 3200（保持不变）

扇区数/磁道 = X（待求）

总容量 = 2×3200×X×S

‌

新旧工艺总容量相等，因此：

4×3200×128×S = 2×3200×X×S

X = 256

新工艺要求每个磁道有‌256个扇区‌。

9. 请写出磁盘扇区访问时间的计算公式，并简单解释寻道时间、旋转时间、传送时间。

10. 假定某计算机的cache采用直接映射方式，和主存交换的数据块大小为1个字，按字编址，一共能存放16个字的数据。CPU开始执行某程序时，cache为空，在该程序执行过程中，CPU依次访问以下地址序列：2、3、11、16、21、13、64、48、19、11、3、22、4、27、6和11。

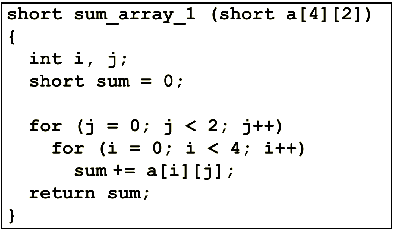
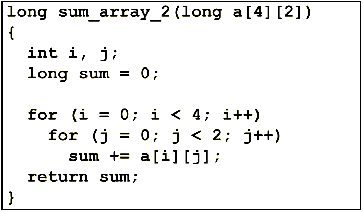
（1）每次访问在cache中命中还是缺失？试计算访问上述地址序列的cache命中率。1/16

1/16

（2）若cache容量还是16个字，而数据块大小改为4个字，则上述地址序列的命中情况又如何？说明块大小和命中率的关系。1/4

4/16

11. 假定某计算机的数据cache采用全相连映射方式，即内存的任意一个块可以被放置到cache中的任意位置。主存与cache交换的块大小为2个字（代码1为2B，代码2为8B），数据cache一共可以存放4个字。当cache发生冲突时，选取最近最久不使用（LRU）的那个块被换出。数组按行优先方式存储，请分别计算代码段1和代码段2的Cache命中率，讨论哪一个代码对cache更友好，并分析cache不友好代码的原因。

代码1 代码2

代码1 的命中率为0，代码2 的命中率为50%，

代码1

1. Cache配置：

* 块大小：2B（可存放1个short元素，因为short占2B）
* Cache总容量：4B（可存放2个块）
* 替换策略：LRU
* 数组元素类型：short（2B）

1. 数组内存布局（行优先）：a[0][0],a[0][1],a[1][0],a[1][1],a[2][0],a[2][1],a[3][0],a[3][1]
2. 访问模式分析：代码采用列优先访问（外层循环j，内层循环i），这与行优先存储不匹配。
3. 访问过程：初始状态：Cache为空

访问a[0][0]：未命中，加载块0（a[0][0]）  
访问a[1][0]：未命中，加载块1（a[1][0]）  
访问a[2][0]：未命中，替换块0（LRU），加载块2（a[2][0]）  
访问a[3][0]：未命中，替换块1（LRU），加载块3（a[3][0]）  
访问a[0][1]：未命中，替换块2（LRU），加载块4（a[0][1]）  
访问a[1][1]：未命中，替换块3（LRU），加载块5（a[1][1]）  
访问a[2][1]：未命中，替换块4（LRU），加载块6（a[2][1]）  
访问a[3][1]：未命中，替换块5（LRU），加载块7（a[3][1]）

1. 命中率：0%
2. Cache不友好原因：

* 访问模式与存储模式不匹配（列访问 vs 行存储）
* cache容量过小，无法容纳足够的行数据

1. 优化建议：

* 改为行优先访问（交换循环顺序）
* 增加cache容量

代码2

Cache配置分析：

* 块大小：2个字（即每次从主存加载2个连续的字）
* Cache总容量：4个字（即可以存放2个块）
* 替换策略：LRU（最近最久未使用）

数组访问模式：

* 数组a[4][2]在内存中的布局（按行优先）：
* a[0][0], a[0][1], a[1][0], a[1][1], a[2][0], a[2][1], a[3][0], a[3][1]

访问a[0][0]：未命中，加载块0（a[0][0],a[0][1]）

访问a[0][1]：命中

访问a[1][0]：未命中，加载块1（a[1][0],a[1][1]）

访问a[1][1]：命中

访问a[2][0]：未命中，替换块0（LRU）

访问a[2][1]：命中

访问a[3][0]：未命中，替换块1（LRU）

访问a[3][1]：命中

命中率仍为50%（4/8）