ICS 第三章

【操作数格式、数据传送】

1．判断下列 x86-64 ATT 操作数格式是否合法。

(1) (　　) 8(%rax, ,2)

(2) (　　) $30(%rax,%rax,2)

(3) (　　) 0x30

(4) (　　) 13(,%rdi,4)

(5) (　　) (%rsi,%rdi,6)

(6) (　　) %ecx

(7) (　　) (%ecx)

(8)★(　　) (%rbp,%rsp)

【答】1 不正确；2 不正确，$；3 正确，是内存地址 0x30；4 正确；5 不正确，缩放比例只能是 1，2，4，8；6 正确；7 不正确，x86-64 不允许将除了 64 位寄存器以外的寄存器作为寻址模式基地址；8 不正确，%rsp 不能作为操作数（参考 Intel 手册 Vol.1 3-23 与 Vol.2A 2-7）。

2．假设%rax、%rbx 的初始值都是 0。根据下列一段汇编代码，写出每执行一步后两个寄存器的值。

	%rax	%rbx
movabsq $0x0123456789ABCDEF, %rax		
---->	0x0123456789ABCDEF	0x0000000000000000
movw %ax, %bx		
---->	0x0123456789ABCDEF	(1) ?????????????
movswq %bx, %rbx		
---->	0x0123456789ABCDEF	(2) ?????????????
movl %ebx, %eax		
---->	(3) ?????????????	(2) ?????????????
Movabsq $0x123456789ABCDEF, %rax		
---->	0x0123456789ABCDEF	(2) ?????????????
cltq		
---->	(4) ?????????????	(2) ?????????????

(1) 0x000000000000CDEF

(2) 0xFFFFFFFFFFFFCDEF

(3) 0x00000000FFFFCDEF

(4) 0xFFFFFFFF89ABCDEF

3．下列操作不等价的是(　　)
A．movzbq 和 movzbl
B．movzwq 和 movzwl
C．movl 和 movslq
D．movslq %eax, %rax 和 cltq

【答】C;　A．B．movzbl/movzwl 生成了四字节，把高位设为 0；　D．cltq 是对%eax 的符号拓展；　C．movl 和 movzlq 等价（所以不需要 movzlq 这条指令）

4．判断下列 x86-64 ATT 数据传送指令是否合法。
(1)　(　　) movl $0x400010, $0x800010
(2)　(　　) movl $0x400010, 0x800010
(3)　(　　) movl 0x400010, 0x800010
(4)　(　　) movq $-4, (%rsp)
(5)　(　　) movq $0x123456789AB, %rax
(6)　(　　) movabsq $0x123456789AB,%rdi
(7)★(　　) movabsq $0x123456789AB,16(%rcx)
(8)★(　　) movq 8(%rsp),%rip

【答】1 不正确，目的不能是立即数；2 正确；3 不正确，两个操作数不能同时是内存地址；4 正确；5 不正确，这里要用 movabsq；6 正确；7 不正确，movabsq 的目标地址必须是整数寄存器；8 不正确，不能用 mov 向%rip 中传入数据。

【加载有效地址、算术运算】

5．在 32 位机器下，假设有如下定义
int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
某一时刻,%ecx 存着第一个元素的地址,%ebx 值为 3,那么下列操作中,哪一个将 array[3]移入了%eax? (　　)

A．leal 12(%ecx), %eax
B．leal (%ecx,%ebx,4), %eax
C．movl (%ecx,%ebx,4), %eax
D．movl 8(%ecx,%ebx,2), %eax

【答】C

6．将下列汇编代码翻译成 C 代码

func: 1 movq %rdi, %rax 2 salq $4, %rax 3 subq %rdi, %rax 4 movq %rax, %rdi 5 leaq 0(,%rsi,8), %rax 6 subq %rsi, %rax 7 addq %rdi, %rax 8 ret	*// a in %rdi, b in %rsi* long func(long a, long b) { return _____; }

【答】a * 15 + b * 7：第 1 行%rax = a；第 2 行%rax = 16a 第 3 行%rax = 15a；第 4 行%rax = %rdi = 15a；第 5 行%rax = 8b；第 6 行%rax = 7b；第 7 行%rax = 15a + 7b

【条件码】

7．指令 setg %al 会让寄存器%al 得到(　　)

A. ~(SF ^ OF) & ~ZF

B. ~(SF | OF) & ~ZF

C. ~(SF | OF)

D. ~(SF ^ OF)

【答】A;　如果 a > b，那么 a - b 要不为正，要不发生上溢变负；同时 a 不等于 b。

8．将下列汇编代码翻译成 C 代码

func: movl %ecx, %r8d subl %edx, %r8d leal (%rcx,%rdx,2), %eax sete %dl movzbl %dl, %edx addl %edx, %eax ret	*// a in %rdi, b in %rsi* int func(int a, int b) { int c = a - b; int d = _____; return _____; }

【答】a + 2 * b; (c == 0) + d；注意 leal 不改变条件码，而 subl 改变 ZF 位，（备注：imul 不改变 ZF 位）

【条件分支】

9．将下列汇编代码翻译成 C 代码

func: movl $1, %eax jmp .L2 .L4: testb $1, %sil je .L3 imulq %rdi, %rax .L3: sarq %rsi imulq %rdi, %rdi .L2: testq %rsi, %rsi jg .L4 rep ret	*// a in %rdi, b in %rsi* long func(long a, long b) { long ans = _____; while (_____) { if (_____) ans = _____; b = _____; a = _____; } return ans; }

【答】

```
long func(long a, long b) {
   long ans = 1;
   while (b > 0) {
      if (b & 1)
         ans = ans * a;
      b = b >> 1;
      a = a * a;
   }
   return ans;
}
```

10．对于下列四个函数，假设 gcc 开了编译优化，判断 gcc 是否会将其编译为条件传送。

long f1(long a, long b) { return (++a > --b) ? a : b; }	Y N
long f2(long *a, long *b) { return (*a > *b) ? --(*a) : (*b)--; }	Y N
long f3(long *a, long *b) { return a ? *a : (b ? *b : 0); }	Y N
long f4(long a, long b) { return (a > b) ? a++ : ++b; }	Y N

【答】f1 由于比较前计算出的 a 与 b 就是条件传送的目标，因此会被编译成条件传送；f2 由于比较结果会导致 a 与 b 指向的元素发生不同的改变，因此会被编译成条件跳转；f3 由于指针 a 可能无效，因此会被编译为条件跳转；f4 会被编译成条件传送，注意到 a 和 b 都是局部变量，return 的时候对 a 和 b 的操作都是没有用的。使用-O1 可以验证 gcc 的行为。

11．使用 GDB 查看某个可执行文件，发现其一段内存为：

```
0x400598: 0x0000000000400488 0x0000000000400488
0x4005a8: 0x000000000040048b 0x0000000000400493
0x4005b8: 0x000000000040049a 0x0000000000400482
0x4005c8: 0x000000000040049a 0x0000000000400498
```

将主函数的汇编代码转换成 C 代码：

```
0x400474: cmp    $0x7, %edi
0x400477: ja     0x40049a
0x400479: mov    %edi, %edi
0x40047b: jmpq   *0x400598(,%rdi,8)
0x400482: mov    $0x15213, %eax
0x400487: retq
0x400488: sub    $0x5, %edx
0x40048b: lea    0x0(,%rdx,4), %eax
0x400492: retq
0x400493: mov    $0x2, %edx
0x400498: and    %edx, %esi
0x40049a: lea    0x4(%rsi), %eax
0x40049d: retq
```

```
// a in %rdi, b in %rsi, c in %rdx
int main (int a, int b, int c) {
    int res = 4;
    switch(a){
        case 0:
        case 1:
            __________;
        case ___:
            res = __________;
            break;
        case ___:
            res = __________;
            break;
        case 3:
            __________;
        case 7:
            __________;
        default:
            __________;
    }
    return res;
}
```

【答】

```
case 0:
case 1:
    c = c - 5;
case 2:
    res = 4 * c; //or res *= c
    break;
case 5:
    res = 86547; //or 0x15213
    break;
case 3:
    c = 2;
case 7:
    b = b & c;
default:
    res += b; // or res = b + 4
```

12．根据汇编指令补充机器码中缺失的字节。

loop:	
4004d0: 48 89 f8	mov %rdi, %rax
4004d3: eb __	jmp 4004d8 <loop+0x8>
4004d5: 48 d1 f8	sar %rax
4004d8: 48 85 c0	test %rax, %rax
4004db: 7f __	jg 4004d5 <loop+0x5>
4004dd: f3 c3	repz retq

【答】03; f8．程序进行到第五行时，开始跳转，跳转位置为 0xf8(-8) + 0x4004dd = 0x4004d5，注意当程序走到第五行进行跳转之前，PC 指向该指令的下一条指令，即第六行 repz 的开头

【过程调用】

13．将下列汇编代码翻译成 C 代码

```
func:
    movq   %rsi, %rax
    testq  %rdi, %rdi
    jne .L7
    rep ret
.L7:
    subq   $8, %rsp
    imulq  %rdi, %rax
    movq   %rax, %rsi
    subq   $1, %rdi
    call   func
    addq   $8, %rsp
    ret
```

```
long func(long n, long m) {
    if (__________)
        return _________;
    return func (_____, _____);
}
```

【答】n == 0; m; n - 1, m * n

14．将下列 C 代码翻译为汇编代码

```
void callee(long *a, long *b) {
    if (a == b)
        return;
    *a ^= *b;
    *b ^= *a;
    *a ^= *b;
    return;
}

void caller(long n, long arr[]) {
    for (long i = 0; i < n/2; i++)
        callee(&arr[i], &arr[n-i]);
}
```

```
callee:
    cmpq   %rsi, %rdi
    je  .L1
    movq   (%rsi), %rax
    xorq   (%rdi), %rax
    movq   %rax, (%rdi)
    xorq   (%rsi), %rax
    movq   %rax, (%rsi)
```

```
caller:
    pushq  %r12
    pushq  %rbp
    pushq  %rbx
    movq   %rdi, %rbp
    movq   %rsi, %r12
    movl   $0, %ebx
    jmp .L4
```

```
    xorq   %rax, (%rdi)
.L1:
    rep ret
```

```
.L5:
    movq   %rbp, %rax
    subq   %rbx, %rax
    leaq   (%r12,%rax,8), %rsi
    leaq   (%r12,%rbx,8), %rdi
    call   callee
    addq   $1, %rbx
.L4:
    movq   %rbp, %rax
    shrq   $63, %rax
    addq   %rbp, %rax
    sarq   %rax
    cmpq   %rbx, %rax
    jg  .L5
    popq   %rbx
    popq   %rbp
    popq   %r12
    ret
```

在 x86-64、操作系统为 Linux 的情况下，假设 main 在 0x4000ac 处调用 caller，caller 在 0x400088 处调用 callee；调用函数（call xx）的代码长度为 5。在 main 即将调用 caller 时，部分寄存器的情况见下表左侧。请在下图右侧画出控制流第一次走到.L1 时，堆栈的结构。

寄存器	调用前的值	地址	内容（不确定的空格填-）
%rsp	0xffffffffff80	0xf...f88~8f	-
%rax	0x0	0xf...f80~87	-
%rbx	0x15	0xf...f78~7f	0x4000b1
%rbp	0x18	0xf...f70~77	0x213
%r12	0x213	0xf...f68~6f	0x18
%rsi	0x0	0xf...f60~67	0x15
%rdi	0x0	0xf...f58~5f	0x40008d
		0xf...f50~57	-

【答】注意不要弄错 Return Address 的内容。

【结构与联合】

15．在 x86-64、Linux 操作系统下有如下 C 定义：

```
struct A {
    char CC1[6];
    int II1;
    long LL1;
    char CC2[10];
    long LL2;
    int II2;
};
```

(1) sizeof(A) = _____字节。
(2) 将 A 重排后，令结构体尽可能小，那么得到的新的结构体大小为_____字节。

【答】56，40；CC1：0；II1：8，必须 4 字节对齐；LL1：16，必须 8 字节对齐；CC2：24；LL2：40，必须 8 字节对齐；II2：48。基本元素最大的为 long，因此 sizeof(A)是 56。重排顺序：LL1 LL2 II1 II2 CC1 CC2，刚好没有空白空间，得到的大小为 8+8+4+4+10+6=40 字节。

16．在 x86-64、LINUX 操作系统下，考虑如下的 C 定义：

```
typedef union {
    char c[7];
    short h;
} union_e;

typedef struct {
    char d[3];
    union_e u;
    int i;
} struct_e;

struct_e s;
```

回答如下问题：

(1) s.u.c 的首地址相对于 s 的首地址的偏移量是_____字节。

(2) sizeof(union_e) = _____字节。

(3) s.i 的首地址相对于 s 的首地址的偏移量是_____字节。

(4) sizeof(struct_e) = _____字节。

(5) 若只将 i 的类型改成 short，那么 sizeof(struct_e) = _____字节。

(6) 若只将 h 的类型改成 int，那么 sizeof(union_e) = _____字节。

(7) 若将 i 的类型改成 short、将 h 的类型改成 int，那么 sizeof(union_e) = _____字节，sizeof(struct_e) = _____字节。

(8) 若只将 short h 的定义删除，那么(1)~(4)问的答案分别是___，___，___，___。

【答】

4; 8; 12; 16; 14; 8; 8, 16; 3, 7, 12, 16。具体解释如下：

(1)~(4)：

d1	d2	d3		h							i				
				c1	c2	c3	c4	c5	c6	c7					

(5)：

d1	d2	d3		h								i	
				c1	c2	c3	c4	c5	c6	c7			

(6)：

d1	d2	d3		h							i				
				c1	c2	c3	c4	c5	c6	c7					

(7)：

d1	d2	d3		h							i			
				c1	c2	c3	c4	c5	c6	c7					

对于(7)，虽然和(5)很像，并且 sizeof(union_e)也没变，但是实际上对齐要求的是所有基本元素都要对齐。所以在考虑 sizeof 的时候，不妨假设开辟一个包含两个元素的结构体数组，检查一下第二个结构体是否对其了所有的基本元素。

(8)：

d1	d2	d3	c1	c2	c3	c4	c5	c6	c7			i

【调试工具】

17．写出使用 gcc 编译源代码 lab.c、生成可执行文件 lab、采用二级编译优化的命令。

【答】gcc lab.c -o lab -O2

18．写出使用 gcc 编译源代码 foo.c、生成汇编语言文件 foo.s 的命令

【答】gcc foo.c -S

19．将可执行文件 bar 逆向工程为汇编代码的工具是(　　)
A．gcc　　B．gdb　　C．objdump　　D．hexedit　　E．gedit

【答】C

20．gdb 中，单步指令执行的命令是
A．r　　B．b　　C．p　　D．finish　　E．si　　F．disas

【答】E

【综合】

21．以下提供了一段代码的 C 语言、汇编语言以及运行到某一时刻栈的情况

汇编：

```
0000000000400596 <func>:
  400596: sub    $0x28,%rsp
  40059a: mov    %fs:0x28,%rax
  4005a3: mov    %rax,0x18(%rsp)
  4005a8: xor    %eax,%eax
  4005aa: mov    (%rdi),%rax
  4005ad: mov    0x8(%rdi),%rdx
  4005b1: cmp    %rdx,%rax
  4005b4: jge    (1)
  4005b6: mov    %rdx,(%rdi)
  4005b9: mov    %rax,0x8(%rdi)
  4005bd: mov    0x8(%rdi),%rax
  4005c1: test   %rax,%rax
  4005c4: jne    4005cb <func+0x35>
  4005c6: mov    (%rdi),%rax
  4005c9: jmp    (2)
  4005cb: mov    (%rdi),%rdx
  4005ce: sub    %rax,%rdx
  4005d1: mov    %rdx,(%rsp)
  4005d5: mov    %rax,0x8(%rsp)
  4005da: mov    (3)          ,%rdi
  4005dd: callq  400596 <func>
  4005e2: mov    0x18(%rsp),%rcx
  4005e7: xor    (4)          ,%rcx
  4005f0: (5)    4005f7 <func+0x61>
  4005f2: callq  400460 <__stack_chk_fail@plt>
  4005f7: add    (6)          ,%rsp
  4005fb: retq

00000000004005fc <main>:
  4005fc: sub    $0x28,%rsp
  400600: mov    %fs:0x28,%rax
  400609: mov    %rax,0x18(%rsp)
  40060e: xor    %eax,%eax
  400610: movq   0x69,(%rsp)
  400618: movq   0xfc,0x8(%rsp)
  400621: mov    %rsp,%rdi
  400624: callq  400596 <func>
  400629: mov    %rax,%rsi
```

```
  40062c: mov    $0x4006e4,%edi
  400631: mov    $0x0,%eax
  400636: callq  400470 <printf@plt>
  40063b: mov    0x18(%rsp),%rdx
  400640: xor    (4)           ,%rdx
  400649: (5)    400650 <main+0x54>
  40064b: callq  400460 <__stack_chk_fail@plt>
  400650: mov    $0x0,%eax
  400655: add    (6)           ,%rsp
  400659: retq
```

C 语言与堆栈：

```
typedef struct{
    long a;
    long b;
} pair_type;
long func(pair_type *p) {
    if (p -> a < p -> b) {
        long temp = p -> a;
        p -> a = p -> b;
        p -> b = temp;
    }
    if ((7)           ) {
        return p -> a;
    }
    pair_type np;
    np.a = (8)           ;
    np.b = (9)           ;
    return func(&np);
}
int main(int argc, char* argv[]) {
    pair_type np;
    np.a = (10)          ;
    np.b = (11)          ;
    printf("%ld", func(&np));
    return 0;
}
```

.....
0x0000000000000000
0xc76d5add7bbeaa00
0x00007fffffffdf60
(a)
(b)
0x0000000000400629
(c)
(d)
0x0000000000000001
0x0000000000000069
0x0000000000000093
(e)
0x00000000ff000000
(f)
0x0000000000000000
(g)
(h)
(i)
0x0000000000000000
(j)
(k)
0x000000000000002a
0x000000000000003f
0x00000000004005e2
（栈顶）[低地址]

一些可能用到的字符的 ASCII 码表：

换行	空格	"	%	(	)	,	0	A	a
0x0a	0x20	0x22	0x25	0x28	0x29	0x2c	0x30	0x41	0x61

回答问题：

I. gdb 下使用命令 x/4b 0x4006e4 后（即查看 0x4006e4 开始的 4 个字节，用 16 进制表示）得到的输出结果是

0x4006e4: 0x_____ 0x_____ 0x_____ 0x_____

II. 互相翻译 C 语言代码和汇编代码，补充缺失的空格（标号相同的为同一格）。

(1) _____<func+____>___

(2) _____<func+____>___

(3) _____

(4) _____

(5) _____

(6) _____

(7) _____

(8) _____

(9) _____

(10)_____

(11)_____

III. 补充栈的内容。使用 16 进制，可以不写前导多余的 0；对于给定已知条件后仍无法确定的值，填写“不确定”；已知程序运行过程中寄存器%fs 的值没有改变。

(a) _____

(b) _____

(c) _____

(d) _____

(e) _____

(f) _____

(g) _____

(h) _____

(i) _____

(j) _____

(k) _____

IV. 程序运行结果为_____。

以下提供了一段代码的 C 语言、汇编语言以及运行到某一时刻栈的情况

汇编代码：

```
0000000000400596 <func>:
  400596:  sub    $0x28,%rsp
  40059a:  mov    %fs:0x28,%rax
  4005a3:  mov    %rax,0x18(%rsp)
  4005a8:  xor    %eax,%eax
  4005aa:  mov    (%rdi),%rax
  4005ad:  mov    0x8(%rdi),%rdx
  4005b1:  cmp    %rdx,%rax
  4005b4:  jge    4005bd <func+0x27>
  4005b6:  mov    %rdx,(%rdi)
  4005b9:  mov    %rax,0x8(%rdi)
  4005bd:  mov    0x8(%rdi),%rax
  4005c1:  test   %rax,%rax
  4005c4:  jne    4005cb <func+0x35>
  4005c6:  mov    (%rdi),%rax
  4005c9:  jmp    4005e2 <func+0x4c>
  4005cb:  mov    (%rdi),%rdx
  4005ce:  sub    %rax,%rdx
  4005d1:  mov    %rdx,(%rsp)
  4005d5:  mov    %rax,0x8(%rsp)
  4005da:  mov    %rsp,%rdi
  4005dd:  callq  400596 <func>
  4005e2:  mov    0x18(%rsp),%rcx
  4005e7:  xor    %fs:0x28,%rcx
  4005f0:  je     4005f7 <func+0x61>
  4005f2:  callq  400460 <__stack_chk_fail@plt>
  4005f7:  add    $0x28,%rsp
  4005fb:  retq

00000000004005fc <main>:
  4005fc:  sub    $0x28,%rsp
  400600:  mov    %fs:0x28,%rax
  400609:  mov    %rax,0x18(%rsp)
  40060e:  xor    %eax,%eax
  400610:  movq   $0x69,(%rsp)
  400618:  movq   $0xfc,0x8(%rsp)
  400621:  mov    %rsp,%rdi
  400624:  callq  400596 <func>
  400629:  mov    %rax,%rsi
  40062c:  mov    $0x4006e4,%edi
  400631:  mov    $0x0,%eax
```

```
  400636:  callq  400470 <printf@plt>
  40063b:  mov    0x18(%rsp),%rdx
  400640:  xor    %fs:0x28,%rdx
  400649:  je     400650 <main+0x54>
  40064b:  callq  400460 <__stack_chk_fail@plt>
  400650:  mov    $0x0,%eax
  400655:  add    $0x28,%rsp
  400659:  retq
```

C 语言与 stack 的情况：

```
typedef struct{
    long a;
    long b;
} pair_type;
long func(pair_type *p) {
    if (p -> a < p -> b) {
        long temp = p -> a;
        p -> a = p -> b;
        p -> b = temp;
    }
    if (p -> b == 0) {
        return p -> a;
    }
    pair_type np;
    np.a = p -> a - p -> b;
    np.b = p -> b;
    return func(&np);
}
int main(int argc, char* argv[]) {
    pair_type np;
    np.a = 105;
    np.b = 252;
    printf("%ld", func(&np));
    return 0;
}
```

.....
0x0000000000000000(u)
0xc76d5add7bbeaa00
0x00007fffffffdf60(u?)
0x0000000000000069
0x00000000000000fc
0x0000000000400629
0x0000000000000000(u)
0xc76d5add7bbeaa00
0x0000000000000001(u)
0x0000000000000069
0x0000000000000093
0x00000000004005e2
0x00000000ff000000(u)
0xc76d5add7bbeaa00
0x0000000000000000(u)
0x000000000000002a
0x0000000000000069
0x00000000004005e2
0x0000000000000000(u)
0xc76d5add7bbeaa00
0x000000ff00000000(u)
0x000000000000002a
0x000000000000003f
0x00000000004005e2
（栈顶）[低地址]

一些可能用到的字符的 ASCII 码表：

换行	空格	"	%	(	)	,	0	A	a
0x0a	0x20	0x22	0x25	0x28	0x29	0x2c	0x30	0x41	0x61

回答问题：

1．gdb 下使用命令 x/4b 0x4006e4 后（即查看 0x4006e4 开始的 4 个字节，用 16 进制表示）得到的输出结果是？

0x4006e4: 0x25 0x6c 0x64 0x00

2．互相翻译 C 语言代码和汇编代码，补充缺失的空格（标号相同的为同一格）。

(1) 4005bd <func+0x27>
(2) 4005e2 <func+0x4c>
(3) %rsp
(4) %fs:0x28
(5) je
(6) $0x28
(7) p -> b == 0
(8) p -> a - p -> b
(9) p -> b
(10) 105
(11) 252

3．补充栈的内容。使用 16 进制，可以不写前导多余的 0；对于给定已知条件后仍无法确定的值，填写“不确定”；已知程序运行过程中寄存器%fs 的值没有改变。

(a) 0x0000000000000069
(b) 0x00000000000000fc
(c) 不确定
(d) 0xc76d5add7bbeaa00
(e) 0x00000000004005e2
(f) 0xc76d5add7bbeaa00
(g) 0x000000000000002a
(h) 0x0000000000000069
(i) 0x00000000004005e2
(j) 0xc76d5add7bbeaa00
(k) 不确定

4．程序输出结果是？

21

【注意】
1 最后一空为 0x00，因为要用\0 作字符串结尾
2(1)(2)较难
2(10)(11)不能颠倒，因为汇编如此写
3(a)(b)两空不能颠倒，因为 func 函数会修改内存的值
程序的作用是用（类似于）更相减损术求最大公约数