**3.1**

|  |  |
| --- | --- |
| 操作数 | 值 |
| %eax | 0x100 |
| 0x104 | 0xAB |
| $0x108 | 0x108 |
| (%eax) | 0xFF |
| 4(%eax) | 0xAB |
| 9(%eax,%edx) | 0x11 |
| 260(%ecx,%edx) | 0x13 |
| 0xFC( ,%ecx,4) | 0xFF |
| (%eax,%edx,4) | 0x11 |

**3.3**

Line 1: %bl不能用作地址寄存器.

Line 2: %ax和movl指令不匹配.

Line 3: 源操作数和目的操作数不能都是内存引用.

Line 4: %sh不是寄存器.

Line 5: 立即数不能是目的操作数.

Line 6: %dx与movl指令不匹配.

Line 7: %si与movb指令不匹配.

**3.6**

|  |  |
| --- | --- |
| 指令 | 结果 |
| leal 6(%eax), %edx | x + 6 |
| leal (%eax,%ecx), %edx | x + y |
| leal (%eax,%ecx,4), %edx | x + 4y |
| leal 7(%eax,%eax,8), %edx | 7 + 9x |
| leal 0xA( ,%eax,4), %edx | 10 + 4x |
| leal 9(%eax,%ecx,2), %edx | 9 + x + 2y |

**3.7**

|  |  |  |
| --- | --- | --- |
| 指令 | 目的 | 值 |
| addl %ecx, (%eax) | 0x100 | 0x100 |
| subl %edx, 4(%eax) | 0x104 | 0xA8 |
| imull $16, (%eax,%edx,4) | 0x10C | 0x110 |
| incl 8(%eax) | 0x108 | 0x14 |
| decl %ecx | %ecx | 0x0 |
| subl %edx, %eax | %eax | 0xFD |

**3.9**

int t1 = y ^ x;

int t2 = t1 >> 3;

int t3 = ~t2;

int t4 = t3 – z;

**3.12**

**A.**

根据mull以及64位的操作, 可判断num\_t是unsigned long long类型.

**B.**

已知y有64位长, 故y可写成如下形式:

y = yh \* 232 + yl (1)

其中yh为y的高32位, yl为y的低32位(下文均以下标h表示高32位, 下标l表示低32位).

因此有:

x \* y = x \* yh \* 232 + x \* yl (2)

因为dest为num\_t型指针, 故我们只需要(2)式的低64位结果.

于是令 s = ( x \* yh)l, t = x \* yl, 有:

\*dest = (s + th) \* 232 + tl (3)

分析过程即为算法, 正确性是显然的.

**3.13**

**A.**

int

**B.**

short

**C.**

unsigned char

**D.**

int, unsigned或指针.

**3.15**

**A.**

8048296

**B.**

8048340

**C.**

804837f

**D.**

80482a4

**E.**

ff 25表示间接跳转指令. 要跳转目标的地址在接下来4个字节中以小端编码表示出来, 即: fc 9f 04 08

**3.16**

**A.**

void goto\_cond(int a, int \*p)

{

if (p == 0)

goto done;

if (a <= 0)

goto done;

\*p += a;

done:

return;

}

**B.**

第一个条件分支是 && 表达式实现的一部分. 如果p为0, 代码会跳过对a > 0的测试.

**3.18**

int test(int x, int y)

{

int val = x ^ y;

if (x < -3)

{

if (x > y)

val = x \* y;

else

val = x + y;

}

else if (x > 2)

val = x - y;

return val;

}

**3.21**

**A.**

%edx被初始化为a + b, 每次循环时都加1, 同时%ecx中的值(a)每次循环也加一.

因此%edx中的值始终等于a + b.

**B.**

|  |  |  |
| --- | --- | --- |
| 寄存器 | 程序值 | 初始值 |
| %ecx | a | a |
| %ebx | b | b |
| %eax | result | 1 |
| %edx | a + b | a + b |

**C.**

Arguments: a at %ebp + 8, b at %ebp + 12

Registers: a in %ecx, b in %ebx, result in %eax, (a + b) in %edx

movl 8(%ebp), %ecx get a

movl 12(%ebp), %ebx get b

movl $1, %eax set result = 1

cmpl %ebx, %ecx compare a:b

jge .L11 if(a>=b) goto done

leal (%eax, %ecx), %edx set %edx = a + b

movl $1, %eax set result = 1

.L12 loop:

imull %edx, %eax result \*= (a+b)

addl $1, %ecx a++

addl $1, %edx (a+b)++

cmpl %ecx, %ebx compare b:a

jg .L12 if(b>a) goto loop

.L11 done:

return result

**D.**

int loop\_while(int a, int b)

{

int result = 1;

if(a >= b)

goto done;

int apb = a + b;

loop:

result \*= apb;

a++;

apb++;

if (b > a)

goto loop;

done:

return result;

}

**3.22**

**A.**

int fun\_a(unsigned x)

{

int val = 0;

while (x)

{

val ^= x;

x >> 1;

}

return val & 1;

}

**B.**

如果x的二进制表示中有奇数个1, 就返回1; 否则, 返回0.

**3.23**

**A.**

int fun\_b(unsigned x)

{

int val = 0;

int i;

for(i = 0; i < 32; i++)

{

val = (x & 1) | (val << 1);

x >> 1;

}

return val;

}

**B.**

将x的二进制串镜像化.

**3.27**

int test(int x, int y)

{

int val = 4 \* x;

if (y > 0)

{

if(x < y)

val = x - y;

else

val = x ^ y;

}

else if (y < -2)

val = x + y;

return val;

}

**3.28**

**A.**

-2, 0, 1, 2, 3, 4.

**B.**

.L6对应2和3.

**3.54**

int decode2(int x, int y, int z)

{

z -= y;

int t = z;

t << 15;

t >> 15;

return (z ^ x) \* t;

}

**3.55**

算法描述见3.12.

汇编代码注释如下:

1 movl 12(%ebp), %esi %esi = xl

2 movl 20(%ebp), %eax %eax = y

3 movl %eax, %edx %edx = y

4 sarl $31, %edx %edx = y >> 31

5 movl %edx, %ecx %ecx = y >> 31

6 imull %esi, %ecx %ecx = xl \* (y >> 31)

7 movl 16(%ebp), %ebx %ebx = xh

8 imull %eax, %ebx %ebx = xh \* y

9 addl %ebx, %ecx %ecx = xl \* (y >> 31) + xh \* y

10 mull %esi %edx = (y \* xl)h, %eax = (y \* xl)l

11 leal (%ecx, %edx), %edx %edx += %ecx

12 movl 8(%ebp), %ecx %ecx = dest

13 movl %eax, (%ecx) 最终结果的低32位在\*dest中

14 movl %edx,4(%ecx) 最终结果的高32位在\*(dest+1)中

**3.56**

**A.**

分别为%esi, %ebx, %edi和%edx.

**B.**

分别为1431655765和-2147483648.

**C.**

mask != 0

**D.**

mask = ( (unsigned) mask >> n)

**E.**

result ^= (mask & x)

**F.**

int loop (int x, int n)

{

int result = 1431655765;

int mask;

for (mask = -2147483648; mask != 0; mask = ( (unsigned) mask )>> n)

result ^= (mask & x);

return result;

}

**3.58**

int switch3(int \*p1, int \*p2, mode\_t action)

{

int result = 0;

switch(action)

{

case MODE\_A:

result = \*p1;

\*p1 = \*p2;

break;

case MODE\_B:

result = \*p1 + \*p2;

\*p2 = result;

break;

case MODE\_C:

\*p2 = 15;

result = \*p1;

break;

case MODE\_D:

\*p2 = \*p1;

result = 17;

case MODE\_E:

result = 17;

break;

default:

result = -1;

}

return result;

}

**3.59**

int switch\_prob(int x, int n)

{

int result = x;

switch(n)

{

case 0x28:

case 0x2a:

result <<= 3; break;

case 0x2b:

result >>= 3; break;

case 0x2c:

result <<= 3;

result -= x;

case 0x2d:

result \*= result;

default:

result += 0x11;

}

return result;

}