作业——第一周

1. 已知某32位整数X，其值为-101（十进制），则其以16进制表示的补码为_____，另一32位整数Y的补码为0xFFFFFF6A，则X+Y的16进制补码(32位)为_____，X-Y的16进制补码为_____。

2. 请证明补码加法公式：$[x]_{补}+[y]_{补} \equiv [x+y]_{补}$（mod 2^w）。 $[*]_{补}$表示整型数据*的补码表示，机器字长为W。

3. 将8位无符号数129转换为8位浮点数（exp域宽度为4 bits, frac域宽度为3bits）

Exp = ?

Frac = ?

4. We are running programs on a machine with the following characteristics:

- Values of type int are 32 bits. They are represented in two's complement （补码）, and they are right shifted arithmetically. Values of type unsigned are 32 bits.
- Values of type float are represented using the 32-bit IEEE floating point format, while values of type double use the 64-bit IEEE floating point format.

We generate arbitrary values x, y, and z, and convert them to other forms as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to other forms */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
double dx = (double) x;
double dy = (double) y;
double dz = (double) z;
```

For each of the following C expressions, you are to indicate whether or not the expression always yields 1.

Expression	Always True?
`(x<y) == (-x>-y)`	Y N
`((x+y)<<4) + y-x == 17*y+15*x`	Y N
`~x+~y+1 == ~(x+y)`	Y N
`ux-uy == -(y-x)`	Y N
`(x >= 0) || (x < ux)`	Y N
`((x >> 1) << 1) <= x`	Y N
`(double)(float) x == (double) x`	Y N
`dx + dy == (double) (y+x)`	Y N
`dx + dy + dz == dz + dy + dx`	Y N

5. In the following questions assume the variables **a and b are signed integers** and that the machine uses two's complement representation. Also assume that MAX_INT is the maximum integer, MIN_INT is the minimum integer, and W is one less than the word length (e.g., W = 31 for 32-bit integers). Match each of the descriptions on the left with a line of code on the right (write in the letter).

   //1's Complement：反码，即按位取反

   //2's Complement：补码

1. One's complement of a

   _____

2. a.

   _____

3. `a & b`.

   _____

4. `a * 7`.

   _____

5. `a / 4` .

   _____

6. `(a < 0) ? 1 : -1` .

   _____

a. `~(~a | (b ^ (MIN_INT + MAX_INT)))`

b. `((a ^ b) & ~b) | (~(a ^ b) & b)`

c. `1 + (a << 3) + ~a`

d. `(a << 4) + (a << 2) + (a << 1)`

e. `((a < 0) ? (a + 3) : a) >> 2`

f. `a ^ (MIN_INT + MAX_INT)`

g. `~((a | (~a + 1)) >> W) & 1`

h. `~((a >> W) << 1)`

i. `a >> 2`

6. 有如下的 C 代码，在 linux X86-64 系统下，生成的汇编代码如有下图，请填上缺失部分。

```
long arith2
(long x, long y, long z)
{
   long t1 = x+z+y;
   long t3 = x+4;
   long t4 = y * 48;
   long t5 = t3 + t4;
   long rval = t1 * t5;
   return rval;
}
```

```
arith2:
        leaq    (%rsi,%rsi,2), %rcx
        addq    %rdi, %rdx
        addq    %rdx, %rsi
        salq    $4,   %rcx
        leaq    4(%rdi,%rcx), %rax
        imulq    %rsi,  [填空]
        ret
```

7. 有如下的 C 语言代码，以及编译生成的对应汇编代码，其中注释掉 if (likely (a == 2))这行生成汇编代码段-1，注释掉 if (unlikely (a == 2)) 这行生成汇编代码段-2。

问题：请简要分析编译指示（directives）
“#define likely(x) __builtin_expect(!!(x), 1)
#define unlikely(x) __builtin_expect(!!(x), 0)”
的作用——为何生成的指令序列的顺序不同，与处理器流水线的运行过程与优化有何关系？

```
#include<stdlib.h>
#define likely(x)    __builtin_expect(!!(x), 1)
#define unlikely(x)  __builtin_expect(!!(x), 0)
int main(char *argv[], int argc)
{
   int a,b;
   /* Get the value from somewhere GCC can't optimize */
   a = atoi (argv[1]);//将字符串转换为 int 整数
   b = a*a;
   if (unlikely (a == 2))
   // if (likely (a == 2))
   {
      a++;  b++;
   }
   else
   {
      a--;  b--;
   }
   return a+b;
}
```

代码段-1

```
main:
        subq    $8, %rsp
        movq    8(%rdi), %rdi
        xorl    %esi, %esi
        movl    $10, %edx
        call    strtol                  # atoi 调用，返回值在 eax 中
        movl    %eax, %esi
        movl    $3, %ecx
        imull   %eax, %esi
        cmpl    $2, %eax
        leal    1(%rsi), %edx
        je      .L3
        leal    -1(%rax), %ecx
        leal    -1(%rsi), %edx
.L3:
        leal    (%rcx,%rdx), %eax
        addq    $8, %rsp
        ret
```

代码段-2

```
main:
        subq    $8, %rsp
        movq    8(%rdi), %rdi
        xorl    %esi, %esi
        movl    $10, %edx
        call    strtol
        movl    %eax, %ecx
        imull   %eax, %ecx
        cmpl    $2, %eax
        jne     .L2
        leal    1(%rcx), %eax
        movl    $3, %edx
.L3:
        addl    %edx, %eax
        addq    $8, %rsp
        ret
.L2:
        leal    -1(%rax), %edx
        leal    -1(%rcx), %eax
        jmp     .L3
```